DECEMBER 1968

SYSTEM AND SOFTWARE SIMULATOR

VOLUME IV

UNITED STATES ARMY
COMPUTER SYSTEMS SUPPORT
AND EVALUATION COMMAND
WASHINGTON, D.C. 20310

3200.8 (Att 1 to Encl 1)
Mar 7, 66

Security Classification

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author) Leo J. Cohen Associates 334 West State Street Trenton, New Jersey	2a. REPORT SECURITY CLASSIFICATION Unclassified 2b. GROUP

3. REPORT TITLE

SYSTEM AND SOFTWARE SIMULATOR VOLUME IV

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

Working Paper

5. AUTHOR(S) (First name, middle initial, last name)

Leo J. Cohen

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
December 1968	424	

8a. CONTRACT OR GRANT NO. DAAB09-68-C-0118 b. PROJECT NO. c. d.	9a. ORIGINATOR'S REPORT NUMBER(S) 9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Distribution of this document is unlimited.

11. SUPPLEMENTARY NOTES This documentation has been assembled and released by the sponsoring activity	12. SPONSORING MILITARY ACTIVITY USA Computer Systems Support and Evaluation Command Washington, D. C.

13. ABSTRACT

The System and Software Simulator (S3) is a digital event simulator written in FORTRAN IV and designed to perform simulations of computer systems hardware and software and of the workload being applied to the system.

This and the other three volumes constitute the complete documentation available on S3. Volume I describes the inputs, outputs, methods and capabilities of S3. Volume II contains the flowcharts, narrative description of the flowcharts, layouts and descriptions of the tables utilized by S3. Volume III contains descriptions of the assembly language used for preparation of input to S3, of the macro capability of the assembler, and of the modifications made to S3 to provide additional output data. Volume IV is the program documentation on the internal workings of the assembler. It consists of table descriptions, flow charts and narratives, and file descriptions.

These volumes are a collection of documentation delivered under two separate contracts. They have not been edited and as such are considered working papers.

DD FORM 1 NOV 65 1473

Security Classification

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Simulation						
Computer Systems						
Operating Systems						
Evaluation						
Timing						
Dynamic Event Simulation						

Security Classification

WORKING PAPER

The documentation on the System and Software Simulator (S3) contained in this and the other three volumes is considered a working paper and no claims are made as to its accuracy. There has been no attempt to edit the information. Discrepancies and inconsistencies are known to exist.

This information is being released as a service to interested parties and to satisfy the numerous requests for information on S3.

The documentation of S3 is contained in four volumes. Volumes I and II are contract end items delivered under contract number DA-49-083 OSA-3306 and contain the technical descriptions of S3. Volume I describes the inputs, outputs, methods and capabilities of S3. Volume II contains the flowcharts, narrative description of the flowcharts, layouts and descriptions of the tables utilized by S3.

Volumes III and IV contain the documentation delivered as contract end items under contract number DAAB09-68-C-0118. Volume III contains descriptions of the assembly language used for preparation of input to S3, of the macro capability of the assembler, and of the modifications made to S3 to provide additional output data. Volume IV is the program documentation on the internal workings of the assembler. It consists of table descriptions, flow charts and narratives, and file descriptions.

WORKING PAPER

WORKING PAPER

USACSSEC S3 Assembler

Programmer's Manual

Presented by

Leo J. Cohen Associates, Inc.
334 West State Street
Trenton, New Jersey
(609) 695-1488

November 8, 1968

WORKING PAPER

TABLE OF CONTENTS

Section I - Table Descriptions

Section II - Flow Charts and Narrative

Section III - File Descriptions

INTRODUCTION

This programmer's manual contains the documentation for the internal workings of the S3 assembler.

The documentation consists of table descriptions, flow charts and narratives, and file descriptions which will be found in Section I, Section II, and Section III respectively.

WORKING PAPER

2

SECTION I

TABLE DESCRIPTIONS

WORKING PAPER

TABLE DESCRIPTIONS

FORTRAN TABLE	DESCRIPTION
IT1 (100)	GENERAL ASSEMBLER TABLE
AT2 (1)-(80)	PRIMARY INPUT AREA (CHAR)
AT2 (81)-(94)	PRIMARY INPUT AREA (WORD)
IT3 (175)	FIX FIELDS OUTPUT AREA
IT4 (800,5)	GLOBAL DICTIONARY
IT5 (800,5)	LOCAL DICTIONARY
IT6 (90)	GLOBAL HASH TABLE
IT7 (90)	LOCAL HASH TABLE
IT8 (175)	MACRO PROTOTYPE AREA
IT9 (14)	CUR OUTPUT AREA
IT10 (9)	SECOND PASS OUTPUT AREA
IT11 (100,9)	SECOND PASS TABLE
IT12 (14)	TITLE AREA
IT13 (44,5)	CPU-DEF TABLE
IT14 (9,30)	MEM-DEF TABLE
IT15 (9,50)	CHAN-DEF TABLE
IT16 (13,50)	DEVICE-DEF TABLE

FORTRAN TABLE	DESCRIPTION
IT17 (3,5)	CPU CONFIGURATION TABLE
IT18 (8,30)	MEMORY CONFIGURATION TABLE
IT19 (8,50)	CHANNEL CONFIGURATION TABLE
IT20 (12,50)	CONTROL CONFIGURATION TABLE
IT21 (13,100)	DEVICE CONFIGURATION TABLE
IT22 (16)	TO-FROM TABLE
IT23 (5,30)	QTABLE
IT24 (8,150)	REAL FILE TABLE
IT25 (80)	INPUT-LEFT JUSTIFIED BY CHARACTER
IT26 (5,16)	LOAD CLASS TABLE
IT27 (5,6)	RUN CLASS TABLE
IT28 (25)	TABLE DUMP CONTROL TABLE
IT29 (10)	STATISTICS CONTROL TABLE
IT30 (3,100)	PROGRAM DISTRIBUTION TABLE
IT31 (22,5)	INTERRUPT VECTOR TABLE
IT32 (3,150)	OP-CODE TABLE
IT33 (3,100)	JOB NAME TABLE
IT34X (4,200)	ORDINAL FILE NAME TABLE
IT35 (10)	O/S MEMORY ALLOCATION TABLE

ERROR CODE RANGES

100	ASM1 - PASS 1
200	PLSYM, GLSYM, PGSYM, GGSYM
300	ASM2 - PASS 2
400	HARDWARE DEFINITIONS
500	FIX FIELDS
600	SYSTEM PARAMETERS

TABLE 1 - GENERAL ASSEMBLER TABLE

Table 1 contains all one word variables which are referenced in two or more subroutines.

A description of each field in table 1 can be found starting on the next page.

GENERAL ASSEMBLER TABLE

IT1(1) ISW1

SUBROUTINE SA4 (FIX FIELDS ROUTINE)

RETURN CODE

1 = COMMENTS CARD

2 = NORMAL STATEMENT

3 = CONTINUATION REQUIRED

IT1(2) ISW2

MACRO DEFINITION SWITCH

0 = NO MACRO DEFINITION IN PROGRESS

1 = MACRO DEFINITION IN PROGRESS

IT1(3) MACNBR

CURRENT MACRO SUFFIX VALUE

IT1(4) N

CURRENT INPUT CHARACTER BEING EXAMINED

IT1(5) CPSW

OPERAND SWITCH

0 = NO MORE OPERANDS

1 = ADDITIONAL OPERANDS

IT1(6) <u>STNBR</u>

CURRENT STATEMENT NUMBER

IT1(7) <u>MIN</u>

MACRO INPUT SW

0 = NOTHING ON MACRO INPUT FILE

1 = DATA ON MACRO INPUT FILE

IT1(8) <u>NGF</u> INITIAL VALUE = 1

NUMBER OF NEXT FREE ENTRY IN LOCAL SYMBOL TABLE

IT1(9) <u>NLF</u> INITIAL VALUE = 1

NUMBER OF NEXT FREE ENTRY IN LOCAL SYMBOL TABLE

IT1(10) <u>NG</u> INITIAL VALUE = 800

MAXIMUM NUMBER OF ENTRIES IN GLOBAL SYMBOL TABLE

IT1(11) <u>NL</u> INITIAL VALUE = 800

MAXIMUM NUMBER OF ENTRIES IN LOCAL SYMBOL TABLE

IT1(12) <u>CIN</u>

COPY INPUT SWITCH

0 = NOTHING ON COPY INPUT FILE

1 = DATA ON COPY INPUT FILE

IT1(13) <u>PEOF</u>

PRIMARY INPUT (CARD READER)

END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(14) <u>CEOF</u>

COPY END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(15) <u>MEOF</u>

MACRO END OF FILE SWITCH

0 = NO END OF FILE

1 = END OF FILE

IT1(16) <u>ISW3</u>

RESOLVE OP-CD OUTPUT SWITCH

0 = NORMAL OP-CD

1 = SPECIAL OP-CD

2 = UNRESOLVED OR MACRO OP-CD

IT1(17) IASM

ASSEMBLY STATEMENT SWITCH

0 = NOT RECEIVED

1 = RECEIVED

IT1(18) OPCD

CURRENT OPERATION CODE

IT1(19)-IT1(23)

NEXT EXPECTED OP-CD TABLE

IT1(24) BLANKS INITIAL VALUE = 6 BLANKS

IT1(25) CURUNT INITIAL VALUE = 15

UNIT FOR CUR OUTPUT TAPE

IT1(26) SEQCHK

SEQUENCE CHECKING COUNTER FOR SECOND PASS

IT1(27) NWR

NUMBER OF WORKER ROUTINE CURRENTLY BEING PROCESSED

IT1(28) ERRSW

ERROR SWITCH

0 = CURRENT STATEMENT HAS NO ERROR

1 = CURRENT STATEMENT HAS AN ERROR

IT1(29) <u>ILNCT</u>

NUMBER OF LINES LEFT ON CURRENT PAGE

IT1(30) <u>IPGCT</u>

CURRENT PAGE COUNT

IT1(31) <u>MACINP</u> INITIAL VALUE = 10

CURRENT MACRO INPUT FILE

IT1(32) <u>MACOPT</u> INITIAL VALUE = 11

CURRENT MACRO OUTPUT FILE

IT1(33) <u>ICPUD</u>

CPU DEFINITION COUNTER

IT1(34) <u>ICFACT</u>

ADJUSTMENT FACTOR FOR COMPUTE STATEMENTS

IT1(35) <u>IMEND</u>

MEMORY DEFINITION COUNTER

IT1(36) <u>ICHAND</u>

CHANNEL DEFINITION COUNTER

IT1(37) <u>ICHCT</u>

CHANNEL TABLE ENTRY COUNTER

IT1(38) <u>PRTSW</u>

0 = DON'T PRINT SECOND PASS OUTPUT

1 = PRINT SECOND PASS OUTPUT

IT1(39) <u>JSTNBR</u>

JOB STATEMENT NUMBER

IT1(40) <u>ICON1</u>

CPU CONFIGURATION COUNTER

IT1(41) <u>ICON2</u>

MEM CONFIG CTR

IT1(42) <u>ICON3</u>

CHAN CONFIG CTR

IT1(43) <u>ICON4</u>

CTL CONFIG CTR

IT1(44) <u>ICON5</u>

DEV CONFIG CTR

IT1(45) <u>ITFP</u>

TO-FROM TABLE IN PROGRESS SW

IT1(46) <u>IROW</u>

CURRENT TO-FROM ROW

IT1(47) <u>INFL</u>

NUMBER TO-FROM FILES

IT1(48) <u>IDEV</u>

DEVICE # FOR CURRENT TO-FROM TABLE

IT1(49) IMER

IT1(50) IQR

QUEUE DEFINITION SWITCH

0 = Q-DEF NOT RECEIVED

1 = Q-DEF RECEIVED

2 = Q-END RECEIVED

IT1(51) IQCTR

QUEUE COUNTER

IT1(52) IQENT

QUEUE ENTRY COUNTER

IT1(53) MULT

COMPUTE MULTIPLICATION FACTOR

IT1(54) NSUM

HASH VALUE OF SYMBOL

IT1(55) IFLR

FILE DEFINITION CONTROL SW

0 = FILES NOT RECEIVED

1 = FILES BEING RECEIVED

2 = FILES RECEIVED

IT1(56) IFLCTR

REAL FILE COUNTER

IT1(57) ILCR

LOAD CLASS CONTROL SW

0 = LOAD CLASS LOT RECEIVED

1 = LOAD CLASS BEING RECEIVED

2 = LOAD CLASS RECEIVED

IT1(58) IRCR

RUN CLASS CONTROL SW

0 = RUN CLASS NOT RECEIVED

1 = RUN CLASS BEING RECEIVED

2 = RUN CLASS RECEIVED

IT1(59) ITAB

TABLE DUMP CONTROL SW

0 = TABLE DUMP CTL NOT RECEIVED

1 = TABLE DUMP CTL RECEIVED

IT1(60) ISTAT

STATISTICS CONTROL SW

0 = STATISTICS CONTROL NOT RECEIVED

1 = STATISTICS CONTROL RECEIVED

IT1(61) KINT

STATISTICS INTERVAL RECEIVED SW

0 = NOT RECEIVED

1 = RECEIVED

IT1(62) JOBSW

JOB SWITCH

0 = NO JOB IN PROGRESS

1 = JOB IN PROGRESS

IT1(63) JOBCTR

JOB COUNTER

IT1(64) OFCTR

ORDINAL FILE COUNTER

IT1(65) MINP

MORE INPUT SWITCH USED IN SMACRO

0 = OFF

1 = ON

IT1(66) ANPS

ANOTHER PASS SWITCH USED IN SMACRO

0 = OFF

1 = ON

IT1(67) HIMAC

HIGH MACRO NBR COUNTER USED IN SMACRO

IT1(68) <u>ERFIL</u>

CURRENT FORTRAN UNIT NUMBER FOR ERROR OUTPUT

IT1(69) <u>OFCTR</u>

ORDINAL FILE COUNTER

IT1(70) <u>KCOUNT</u>

NUMBER OF STATISTICAL INTERVALS FOR SIMULATION

IT1(71) <u>STSW</u>

STORE SWITCH

0 = NO STORE IN PROGRESS

1 = STORE IN PROGRESS

TABLE 2 - PRIMARY INPUT AREA

Table 2 is divided into two sections. The first 80 words, AT2(1) to AT2(80), contain 80 characters as read from the current input source. Each word contains one character left justified and right filled with blanks. The second section of table 2 consists of 14 words, AT2(81) to AT2(94). The first 13 words of this section contain six characters each and the 14th word contains two characters left justified and right filled with blanks.

TABLE 3 - FIXED FIELDS OUTPUT AREA

Table 3 is the area used to store the output from the fix fields routine (SA4, SA5). Word 1 is not normally used. Word 2 contains the total number of operands in this statement in the first 12 bits. Word 2 also contains the two character statement variable, if any, in the last 12 bits. Words 3 and 4 contain up to 12 characters of the statement label. Word 5 contains the length of the statement label in the first 6 bits. The next 12 bits are used to hold a code which indicates if the label is fixed or variable. Labels may be variable only when a macro definition is in progress. The next 12 bits contain a code which indicates if a macro number is to be appended to the label. This code may be set only if a macro definition is currently in progress. Words 6 and 7 contain up to 12 characters of operation code. Word 8 contains the length of the operation code in the first 6 bits. The next 12 bits contain a code

which indicates if the operation code is fixed or variable. Variable operation codes are permitted only in macro definitions. Words 9 and 10 contain up to 12 characters of the first operand for this statement. Word 11 contains the length of the first operand in the first 6 bits. The next 12 bits contain a code which indicates if the first operand is fixed or variable. The next 12 bits indicate whether a macro number should be appended to this operand or not. The remainder of table 3 consists of three word units each of which is used to describe a single operand. A maximum of 55 operands may be accomodated by table 3.

TABLE 3

FIXED FIELDS OUTPUT AREA

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	NOT USED					
2	NUMBER OF OPERANDS				STATEMENT VARIABLE	
3	LABEL PART I					
4	LABEL PART II					
5	LABEL LENGTH	0 = FIXED LABEL N = VARIABLE LABEL		0 = NO MACRO # 99 = ADD MACRO #		0
6	OP-CD PART I					
7	OP-CD PART II					
8	OP-CD LENGTH	0 = FIXED OP-CD 99 = VARIABLE OP-CD				0
9	OPERAND-1 PART I					
10	OPERAND-1 PART II					
11	OPERAND-1 LENGTH	0 = FIXED OPERAND 99 = VARIABLE OPERAND		0 = NO MACRO # 99 = ADD MACRO #		0
⋮						
171	OPERAND 55 PART I					
172	OPERAND 55 PART II					
173	OPERAND-55 LENGTH	0 = FIXED OPERAND 99 = VARIABLE OPERAND		0 = NO MACRO # 99 = ADD MACRO #		0

TABLE 4 - GLOBAL DICTIONARY

The Global Dictionary, table 4, can contain up to 800 five word entries. Each entry is used to describe a single global symbol. The following page describes the various types of symbols which may be found in the global symbol table.

The first two words of a Global Dictionary entry contain up to 12 characters of the symbol. The 3rd word of an entry contains the type of symbol as described on the following page. The 4th word of an entry contains the value which has been assigned to this symbol. The 5th word of an entry may contain a pointer to subsequent entries on this chain in the Global Dictionary. For a brief description of the use of table 4 see the PGSYM, GGSYM, and the HASH subroutine descriptions.

GLOBAL SYMBOL TABLE TYPES

TYPE	DESCRIPTION
1	AT = 1 IOT = 2
2	LIB = 1 NCAT = 2 CAT = 3
3	cpu-name
4	mem-names
5	chan-name
6	IN = 1 OUT = 2 I/O = 3
7	ctl-names
8	SEIZE = 1 NOSEIZE = 2
9	dev-names
10	queue-names
11	NOT USED
12	PRI = 1 FIFO = 2 LIFO = 3

TYPE	DESCRIPTION
13	real-file-names
14	function-names
15	global-equates
16	job-name
17	interrupt-names

TABLE 4

GLOBAL DICTIONARY

WORD		
1	SYMBOL	PART I
2	SYMBOL	PART II
3	SYMBOL	TYPE
4	SYMBOL	VALUE
5	CHAIN	INDEX

TABLE 5 - LOCAL DICTIONARY

The Local Dictionary can contain up to 800 symbols as defined by the worker routine currently in process. Each time an END statement is encountered the current Local Dictionary is written out to FORTRAN unit number 8, and the Local Dictionary table is initialized to contain zeroes. The Local Dictionary File as contained on FORTRAN unit number 8 is then used by the second pass to resolve symbols from the intermediate assembly file.

Words 1 and 2 for an entry in the Local Dictionary can contain up to 12 characters from a local symbol. Word 3 contains the symbol type as described on the following page. Word 4 contains the value assigned to this symbol. Word 5 may contain a pointer to subsequent entries in the current Local Dictionary table. For a complete description of the use of the Local Dictionary see the PLSYM, GLSYM, and HASH subroutine descriptions.

LOCAL SYMBOL TABLE TYPES

TYPE	DESCRIPTION
1	statement-label
2	ordinal-file-name
3	local-equate

TABLE 5

LOCAL DICTIONARY

WORD		
1	SYMBOL	PART I
2	SYMBOL	PART II
3	SYMBOL	TYPE
4	SYMBOL	VALUE
5	CHAIN	INDEX

TABLE 6 - GLOBAL HASH TABLE

The Global Hash Table consists of 90 words, all of which are initialized to zero when the assembler is loaded. When a symbol is to be entered into the global symbol table a value from 1 to 90 is calculated by the HASH subroutine from the characters in the symbol to be placed into the table. That address is then used to place a pointer to an address in the global symbol table into the global hash table. The global hash table then contains pointers to entries in the global symbol table.

TABLE 6

GLOBAL HASH TABLE

WORD	
1	GLOBAL SYMBOL TABLE POINTER
2	GLOBAL SYMBOL TABLE POINTER
	⋮
90	GLOBAL SYMBOL TABLE POINTER

TABLE 7 - LOCAL HASH TABLE

The Local Hash Table consists of 90 words, all of which are initialized to zero when the assembler is loaded. When a symbol is to be entered into the local symbol table a value from 1 to 90 is calculated by the HASH subroutine from the characters in the symbol to be placed into the table. That address is then used to place a pointer to an address in the local symbol table into the local hash table. The hash table then contains pointers to entries in the local symbol table.

TABLE 7

LOCAL HASH TABLE

WORD	
1	LOCAL SYMBOL TABLE POINTER
2	LOCAL SYMBOL TABLE POINTER
90	LOCAL SYMBOL TABLE POINTER

TABLE 8 - MACRO PROTOTYPE AREA

Table 8 is used to store the label, operation code, and operands for macro calls during processing by the SMACRO subroutine. Table 8 is filled by copying the current entry from table 3. Therefore, table 8 is an exact duplicate of table 3.

TABLE 8

MACRO PROTOTYPE AREA

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	NOT USED					
2	NUMBER OF OPERANDS				STATEMENT VARIABLE	
3	LABEL PART I					
4	LABEL PART II					
5	LABEL LENGTH	NOT USED				
6	MACRO NAME					
7	NOT USED					
8	MACRO NAME LENGTH	NOT USED				
9	OPERAND-1 PART I					
10	OPERAND-1 PART II					
11	OPERAND-1 LENGTH	NOT USED				
171	OPERAND 55 PART I					
172	OPERAND 55 PART II					
173	OPERAND-55 LENGTH						

TABLE 9 - CURRENT OUTPUT AREA

Table 9 consists of 14 words used to build records for entry into the PCF library. These entries are then placed on the CUR output tape by means of the CUROUT subroutine. Table 9 is also used to store 14 word records retrieved from the library by means of the SRCHNV and NXTIMT subroutines.

TABLE 10 - SECOND PASS OUTPUT AREA

Table 10 consists of 9 words which are used to build records by the second pass for output to the simulator. The first word of this table contains the current worker routine number. The second word contains the current worker routine statement number. The third word contains the numeric operation code for this statement. Words 4 through 9 contain the numeric values for operands 1 through 6.

TABLE 10

SECOND PASS OUTPUT AREA

WORD	
1	W/R NUMBER
2	W/R STATEMENT NUMBER
3	OP-CODE
4	OPERAND -1
5	OPERAND -2
6	OPERAND -3
7	OPERAND -4
8	OPERAND -5
9	OPERAND -6

TABLE 11 - SECOND PASS TABLE

Table 11 consists of a nine word entry for each operation code which is capable of being used by the simulator. The operation code value is used as an index to table 11. Therefore, operation code 1 would cause the second pass to use entry 1 in table 11. The nine words in each entry in table 11 are used to control the processing performed by the second pass on each statement before it is written out for use by the simulator. Word 1 may contain either a zero or a number indicating a pre-process is required for the current statement. An example of a pre-process would be sequence checking a JOB statement, CF statement, MEM-1 statement, and GEN statement. Word 2 contains a zero or a number indicating that post-processing is necessary for this statement. An example of post-processing would be checking to insure that a SELECT ATQUEUE1 statement referenced a queue which contained ATs.

Word 3 contains the instruction length for the current statement as used by the simulator. This value is used to maintain a current location counter which is printed with each statement if the PRINT option was specified in the ASSEMBLY statement.

Words 4 through 9 may contain a zero, a positive value, or a negative value. These words control the processing for operands 1 through 6 of the current statement. If the operand process number is zero, the operand must be omitted and any operand present with that number will be flagged as an error. If the process number for an operand is positive the operand must be included and the number indicates the process which will be used to convert the operand into an output value. If a process number for an operand is negative, the operand is optional. If the operand is present, the number indicates the process which is to be used to convert the operand to an output value. However, if the operand is missing no error is indicated.

TABLE 11

SECOND PASS TABLE

WORD	
1	PRE-PROCESS NUMBER
2	POST-PROCESS NUMBER
3	S3 INSTRUCTION LENGTH
4	OPERAND-1 PROCESS NUMBER
5	OPERAND-2 PROCESS NUMBER
6	OPERAND-3 PROCESS NUMBER
7	OPERAND-4 PROCESS NUMBER
8	OPERAND-5 PROCESS NUMBER
9	OPERAND-6 PROCESS NUMBER

TABLE 12 - TITLE AREA

Table 12 consists of 14 words which are used to store the page title as specified by the user. Whenever a TITLE statement is encountered the following card is read into table 12. The contents of this table are printed at the top of each page by the PAGE subroutine.

TABLE 13 - CPU DEFINITION TABLE

Table 13 contains a maximum of five 44 word entries, each of which completely defines a CPU. This table is filled as specified by the CPU-DEF statement. If either CAT or NCAT are specified, this table is filled by reading from the current input stream. If LIB is specified this table is filled by obtaining a CPU definition from the library. Each entry in this table is completely described by the following table layout.

TABLE 13

CPU-DEF TABLE

WORD		WORD	
1	CPU-ID PART I	23	E1-A FIXED
2	CPU-ID PART II	24	E1-B POINT
3	CARD CODE	25	E2-A TABLE
4	CPU-ID (CARD)	26	E2-B
5	LOGICAL DATA UNIT	27	E3-A
6	DEC-DIG	28	E3-B
7	DEC. ADD TIME	29	E4-A
8	DEC, MULT TIME	30	E4-B
9	DEC. DIV TIME	31	E1-A FLOATING
10	FIXED ADD TIME	32	E1-B POINT
11	FIXED MULT TIME	33	E2-A TABLE
12	FIXED DIV TIME	34	E2-B
13	FLOAT ADD TIME	35	E3-A
14	FLOAT MULT TIME	36	E3-B
15	FLOAT DIV TIME	37	NOT USED
16	CARD CODE	38	NOT USED
17	CPU-ID	39	NOT USED
18	INST LENGTH	40	NOT USED
19	DEC-DIG/LDU	41	NOT USED
20	CHAR/LDU	42	NOT USED
21	MOVE TIME	43	NOT USED
22	MOVE-E TIME	44	NOT USED

TABLE 14 - MEM-DEF TABLE

The Memory Definition Table may contain up to a maximum of 30 nine word memory definitions. Entries in the memory definition table are filled as specified by the MEM-DEF statement. If CAT or NCAT is specified the memory definition table is filled from the current input stream. If LIB is specified the memory definition table entry is filled on the library. Each word of an entry in the memory definition table is described in the following table layout.

TABLE 14

MEM-DEF TABLE

WORD	
1	MEM-ID PART I
2	MEM-ID PART II
3	CARD CODE
4	MEM-ID (CARD)
5	MEM ACCESS UNIT
6	MEM CYCLE TIME
7	MEM ACCESS TIME
8	MEM SIZE
9	PAGE SIZE

TABLE 15 - CHAN-DEF TABLE

The Channel Definition Table can contain up to a maximum of fifty 9 word channel definitions. Each entry in this table is filled as specified by a CHAN-DEF statement. If CAT or NCAT is specified the current entry is filled from the current input source. If LIB is specified, the current entry is filled from the library. Each word of a Channel Definition entry is described in the following table layout.

TABLE 15

CHAN-DEF TABLE

WORD	
1	CHAN-ID PART I
2	CHAN-ID PART II
3	CARD CODE
4	CHAN-ID (CARD)
5	TYPE
6	SEL OR BURST RATE
7	% CPU INTERFERENCE
8	MPX RATE
9	% CPU INTERFERENCE

TABLE 16 - DEVICE-DEF TABLE

The Device Definition Table is capable of containing up to fifty 13 word device definitions. Each entry in the Device Definition Table is filled as specified by the DEV-DEF statement. If CAT or NCAT is specified, the current entry in the table is filled from the input source. If LIB is specified the current entry in the Device Definition Table is filled from the library. Each word in a device definition table entry is described by the following table layout.

TABLE 16

DEVICE-DEF TABLE

WORD		
1	DEV-ID	PART I
2	DEV-ID	PART II
3	CARD CODE	
4	DEV-ID (CARD)	
5	TYPE	
6	DEV-RATE	
7	TRANSFER WIDTH	
8	START TIME	
9	STOP TIME	
10	REWIND TIME	
11	PRE-PENALTY TIME	
12	PENALTY TIME	
13	FORM TIME	

TABLE 17 - CPU CONFIGURATION TABLE

Table 17 can contain up to five entries, each of which describes a CPU for the current simulation. Each entry in table 17 is filled as specified by the CPU statement. Words 1 and 2 contain up to 12 characters of the CPU-NAME. Word 3 contains a pointer to the CPU definition which describes this CPU.

TABLE 17

CPU CONFIGURATION TABLE

WORD	
1	CPU-NAME PART I
2	CPU-NAME PART II
3	CPU-DEF-INDEX

TABLE 18 - MEMORY CONFIGURATION TABLE

Table 18 can contain the configuration data for up to 30 memories. Each entry in this table is filled from the data supplied with a MEM statement. The first two words contain up to 12 characters of the MEM-NAME. Words 3 through 7 can contain CPU numbers which are used to define the CPUs to which this memory is attached. Word 8 contains the index of the entry in the memory definition table which describes the physical characteristics of this memory. The organization of a single entry in the memory configuration table is described by the following table layout.

TABLE 18

MEMORY CONFIGURATION TABLE

WORD	
1	MEM-NAME PART I
2	MEM-NAME PART II
3	CPU-# INDEX
4	"
5	"
6	"
7	"
8	MEM-DEF INDEX

TABLE 19 - CHANNEL CONFIGURATION TABLE

The Channel Configuration Table can contain the configuration data for up to 50 channels. The information for each entry in this table is extracted from the CHANNEL statement. The first two words of an entry contain up to 12 characters of the CHAN-NAME. Words 3 through 7 can contain CPU numbers which are used to indicate the CPUs to which this channel is attached. Word 8 contains the index of an entry in the channel definition table which describes the physical characteristics of this channel. A complete description of a channel configuration table entry is given by the following table layout.

TABLE 19

CHANNEL CONFIGURATION TABLE

WORD	
1	CHAN-NAME PART I
2	CHAN-NAME PART II
3	CPU-# INDEX
4	"
5	"
6	"
7	"
8	CHAN-DEF INDEX

TABLE 20 - CONTROL CONFIGURATION TABLE

The Control Configuration Table is capable of holding the configuration data for up to 50 control units. Each entry in this table is filled from the information obtained from a CONTROL statement. The first two words of an entry in this table contain up to 12 characters of a CTL-NAME. The third word contains a code which indicates whether this control unit is capable of handling INPUT, OUTPUT, or INPUT and OUTPUT operations. Words 4 through 12 may contain channel numbers indicating those channels to which this control unit is attached. A detailed description of each table entry for the control configuration table is provided by the following table layout.

TABLE 20

CONTROL CONFIGURATION TABLE

WORD	
1	CTL-NAME PART I
2	CTL-NAME PART II
3	I-O-I/O CODE
4	CHAN-# INDEX
5	"
6	"
7	"
8	"
9	"
10	"
11	"
12	"

TABLE 21 - DEVICE CONFIGURATION TABLE

The Device Configuration Table is capable of containing configuration data for up to 100 devices. The configuration data for each entry in the device configuration table comes from a DEVICE statement. The first two words contain up to 12 characters of the DEV-NAME. Word 3 contains a code which indicates if this device is seizable or not. Words 4 through 12 may contain a control number, indicating those control units to which this device is attached. Word 13 contains the number of the entry in the device definition table which describes the physical characteristics of this device. A complete description of an entry in the device configuration table is supplied in the following table layout.

TABLE 21

DEVICE CONFIGURATION TABLE

WORD	
1	DEV-NAME PART I
2	DEV-NAME PART II
3	SEIZE CODE
4	CTL-# INDEX
5	"
6	"
7	"
8	"
9	"
10	"
11	"
12	"
13	DEV-DEF INDEX

TABLE 22 - TO-FROM TABLE

The To-From Table contains enough room to build a single entry in the to-from table. As each entry is built, it is written out to the simulator input file. Word 1 contains the device number for which this to-from table is being built. Word 2 contains the number of the row currently being described. Word 3 contains the dimensions of this to-from array. Words 4 through 16 contain the 13 possible entries in a to-from table. A complete description of a single entry in the to-from table is given by the following table layout.

TABLE 22

TO-FROM TABLE

WORD	
1	DEV-#
2	TO ROW
3	ARRAY DIMENSION
4	ENTRY 1
5	ENTRY 2
6	ENTRY 3
7	ENTRY 4
8	ENTRY 5
9	ENTRY 6
10	ENTRY 7
11	ENTRY 8
12	ENTRY 9
13	ENTRY 10
14	ENTRY 11
15	ENTRY 12
16	ENTRY 13

TABLE 23 - QUEUE TABLE

The Queue Table can contain up to 30 queue definitions. The data for each entry in the queue table is obtained from the QUEUE statement. The first two words contain up to 12 characters of the QUEUE-NAME. The third word contains the maximum number of entries for the queue. The fourth word contains a code which indicates whether this queue can contain ATs or IOTs. Word 5 contains a code to indicate whether this queue should be processed as a FIFO, LIFO, or PRI type queue. A complete description of a single entry in the queue table is provided by the following table layout.

TABLE 23

QUEUE TABLE

WORD	
1	Q-NAME PART I
2	Q-NAME PART II
3	MAX-NO-ENTRIES
4	Q-CONTROL
5	Q-METHOD

TABLE 24 - REAL FILE TABLE

Table 24 can contain up to 150 real file definitions. The data for an entry in the real file table is obtained from the RF or RFC statements. Words 1 and 2 contain up to 12 characters of the real file name. Word 3 contains the number of this real file. Word 4 contains the number of the device on which this real file resides. Word 5 contains the relative location of this file on the device. Word 6 contains the number of characters in a single physical record in this file. Word 7 contains the number of logical records in a single physical record of this file. Word 8 contains the total number of physical records in this file. A complete description of a single entry in the real file table is supplied in the following table layout.

TABLE 24

REAL FILE TABLE

WORD	
1	RF-NAME PART I
2	RF-NAME PART II
3	RF-#
4	DEVICE #
5	RELATIVE LOCATION
6	BUFFER LENGTH (CHAR)
7	RECORDS/BUFFER
8	BUFFERS/FILE

TABLE 25 - INPUT LEFT-JUSTIFIED BY CHARACTER

Table 25 contains 80 characters from the current input record, with one character per word, left justified, and right filled with blanks.

TABLE 26 - LOAD CLASS TABLE

The Load Class Table is capable of containing up to 15 load class entries. The 16th entry is used to temporarily store information for a load class statement in which an error has been found. Each entry in the load class table may contain up to 5 CPU numbers. A complete description of a load class table entry may be found in the following table layout.

TABLE 26

LOAD CLASS TABLE

WORD	
1	CPU-#
2	CPU-#
3	CPU-#
4	CPU-#
5	CPU-#

TABLE 27 - RUN CLASS TABLE

The Run Class Table is capable of containing up to five run class entries. A sixth entry is provided to temporarily store data when a run class description has been found to contain an error. Each entry in the run class table may contain up to 5 CPU numbers. A complete description of a run class table entry may be found in the following table layout.

TABLE 27

RUN CLASS TABLE

WORD	
1	CPU-#
2	CPU-#
3	CPU-#
4	CPU-#
5	CPU-#

TABLE 28 - TABLE DUMP CONTROL TABLE

Table 28 contains 25 words, each of which may be set to control the printing of tables contained within the simulator. If a word contains a zero, the corresponding table in the simulator will be printed at the end of each statistical interval. If a word contains a minus 1, the corresponding table in the simulator will not be printed at the end of a statistical interval. A detailed layout for table 28 may be found on the following page.

TABLE 28

TABLE DUMP CONTROL TABLE

WORD	
1	TI 0=ON -1=OFF
2	T2
3	T3
4	T4
5	T5
6	T6
7	T7
8	T8
9	T9
10	T10
11	T11
12	T12
13	T13

WORD	
14	T14
15	T15
16	T16
17	T17
18	T18
19	T19
20	T20
21	T21
22	T22
23	T23
24	T24
25	T25

TABLE 29 - STATISTICS CONTROL TABLE

The Statistics Control Table contains 10 words, each of which may be used to control the printing of statistics in the simulator at the end of simulation intervals. If a word contains a zero, the corresponding statistics will be printed at the end of each statistical interval. If a word contains a minus one, the corresponding statistics will not be printed at the end of intervals. A complete description of the Statistics Control Table may be found in the following table layout.

TABLE 29

STATISTICS CONTROL TABLE

WORD	
1	ST1 0=ON -1=OFF
2	ST2
3	ST3
4	ST4
5	ST5
6	ST6
7	ST7
8	ST8
9	STAT
10	STAT1

TABLE 30 - PROGRAM DISTRIBUTION TABLE

The Program Distribution Table can contain program distribution information for up to 100 worker routines. This table is normally initialized so that each program is received by CPU 1 and has a load class number of 1. If a program is to be handled in some way other than normal, the RCV statement must be used in that program. The RCV statement may be used to specify the receiving CPU and the load class which is to be used for that program. Word 1 of each entry contains the program number. The second word contains the number of the receiving CPU for that program. Word 3 contains the number of the load class to be used by the current program. A description of a single entry in the program distribution table is supplied by the following table layout.

TABLE 30

PROGRAM DISTRIBUTION TABLE

WORD	
1	PROG #
2	RECEIVING CPU-#
3	LOAD CLASS-#

TABLE 31 - INTERRUPT VECTOR TABLE

The Interrupt Vector Table can contain interrupt vector information for up to 5 CPUs. The first word in an interrupt vector table entry contains the number of the CPU for which this entry applies. The second word contains the number of the operating system to be used by this CPU. Words 3 through 22 contain the statement number for each of the possible 20 interrupts which have been defined.

TABLE 31

INTERRUPT VECTOR TABLE

WORD		WORD	
1	CPU-#	12	INT-10 STMT-#
2	O/S PRCS-#	13	INT-11 "
3	INT-1 STMT-#	14	INT-12 "
4	INT-2 "	15	INT-13 "
5	INT-3 "	16	INT-14 "
6	INT-4 "	17	INT-15 "
7	INT-5 "	18	INT-16 "
8	INT-6 "	19	INT-17 "
9	INT-7 "	20	INT-18 "
10	INT-8 "	21	INT-19 "
11	INT-9 "	22	INT-20 "

TABLE 32 - OPERATION CODE TABLE

The Operation Code Table is capable of containing up to 128 twelve character operation codes with their associated values. This table is loaded by the SOPCD subroutine before the actual reading of input data is begun. This table is loaded from a file catalogued in the library under the name OPCDS/VERS1. These entries in the library must have been processed by ASX001 to insure that they are in alphabetical order. This table is used by subroutine S1 which performs a binary search looking for the current operation code. Words 1 and 2 contain the operation symbol. Word 3 contains the value of the operation code.

TABLE 32

OPERATION CODE TABLE

WORD	
1	OP-SYMBOL PART I
2	OP-SYMBOL PART II
3	VALUE

TABLE 33 - JOB NAME TABLE

The Job Name Table contains the name of each worker routine entered into the system in the sequence in which it appeared in the system. This table is then written out for use by the assembler statistical analysis program. Table 33 can contain up to 100 job names. Words 1 and 2 contain the 12 character job names and word 3 contains the starting ordinal file number for this job. A description of a single entry in the job name table is supplied in the following table layout.

TABLE 33

JOB NAME TABLE

WORD		
1	JOB-NAME	PART I
2	JOB-NAME	PART II
3	STARTING	OF-#

TABLE 34X - ORDINAL FILE NAME TABLE

The Ordinal File Name Table contains the names of all ordinal files utilized in the current run. Words 1 and 2 contain the ordinal file name, word 3 contains the number of the real file to which this ordinal file referred, and word 4 contains the number of buffers to be used in processing this ordinal file. A description of a single entry in the ordinal file name table can be seen in the following table layout.

TABLE 34X

ORDINAL FILE NAME TABLE

WORD	
1	OF-NAME PART I
2	OF-NAME PART II
3	RF-#
4	NUMBER OF BUFFERS

TABLE 35 - O/S MEMORY ALLOCATION TABLE

The operating system memory allocation table can contain 5 pairs of entries which assign up to 5 operating system programs to the memories in which they are to reside. The first word of a pair contains the operating system program number. The second word of a pair contains the number of the memory in which that operating system is to be placed. The data in the O/S memory allocation table comes from the OS statement. A complete description of the O/S memory allocation table may be seen in the following table layout.

TABLE 35

O/S MEMORY ALLOCATION TABLE

WORD	
1	O/S PROG #
2	MEM #
3	O/S PROG #
4	MEM #
5	O/S PROG #
6	MEM #
7	O/S PROG #
8	MEM #
9	O/S PROG #
10	MEM #

WORKING PAPER

86

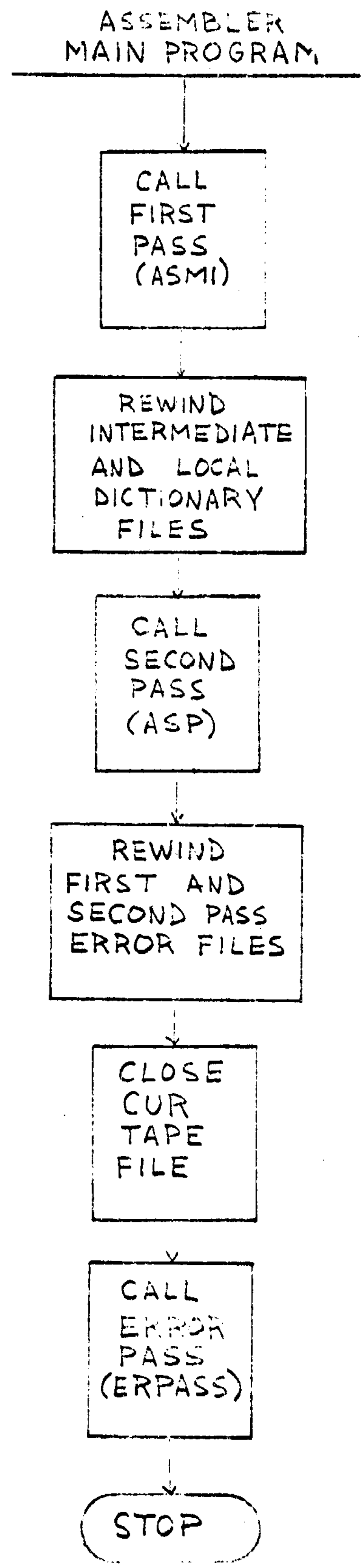
SECTION II

FLOW CHARTS AND NARRATIVE

WORKING PAPER

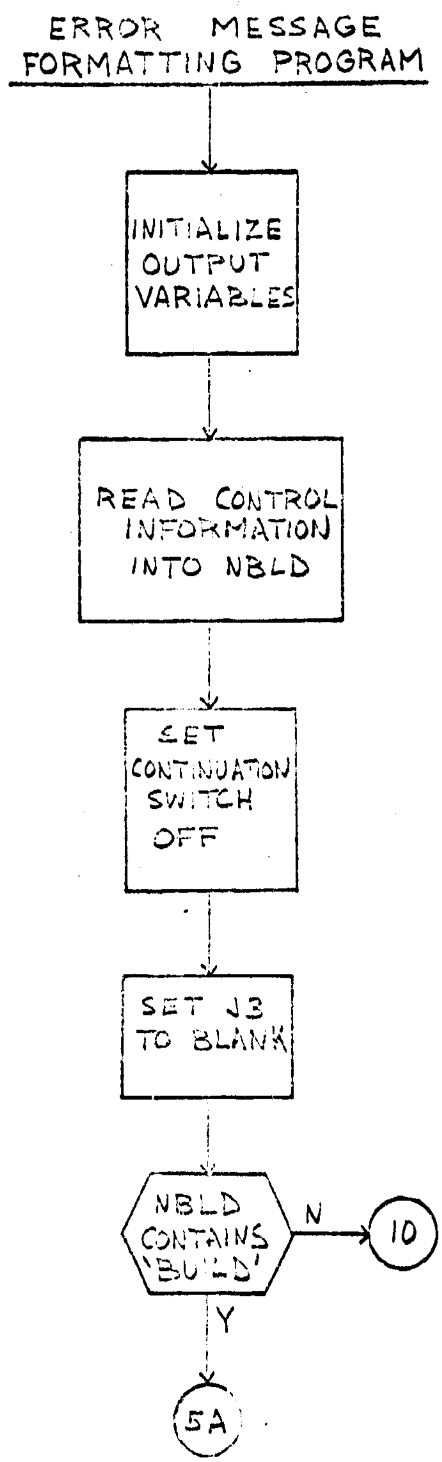
ASSEMBLER MAIN PROGRAM

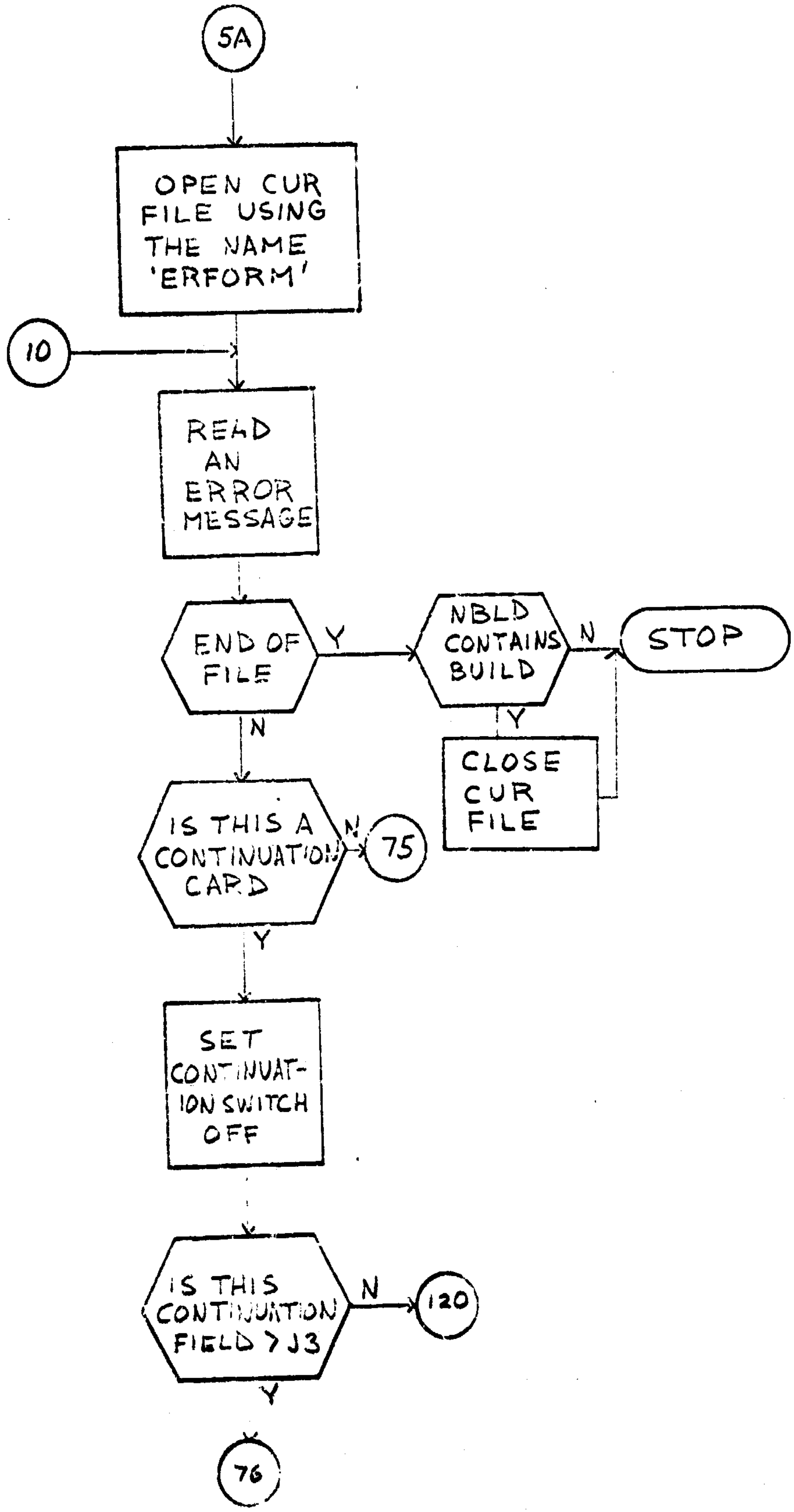
The assembler main program calls the first pass subroutine, ASM1, first. It then rewinds the intermediate and local dictionary files. The second pass, ASP, is then called. The first and second pass error files are then rewound. The CUR tape file is then closed. The error pass is then called.

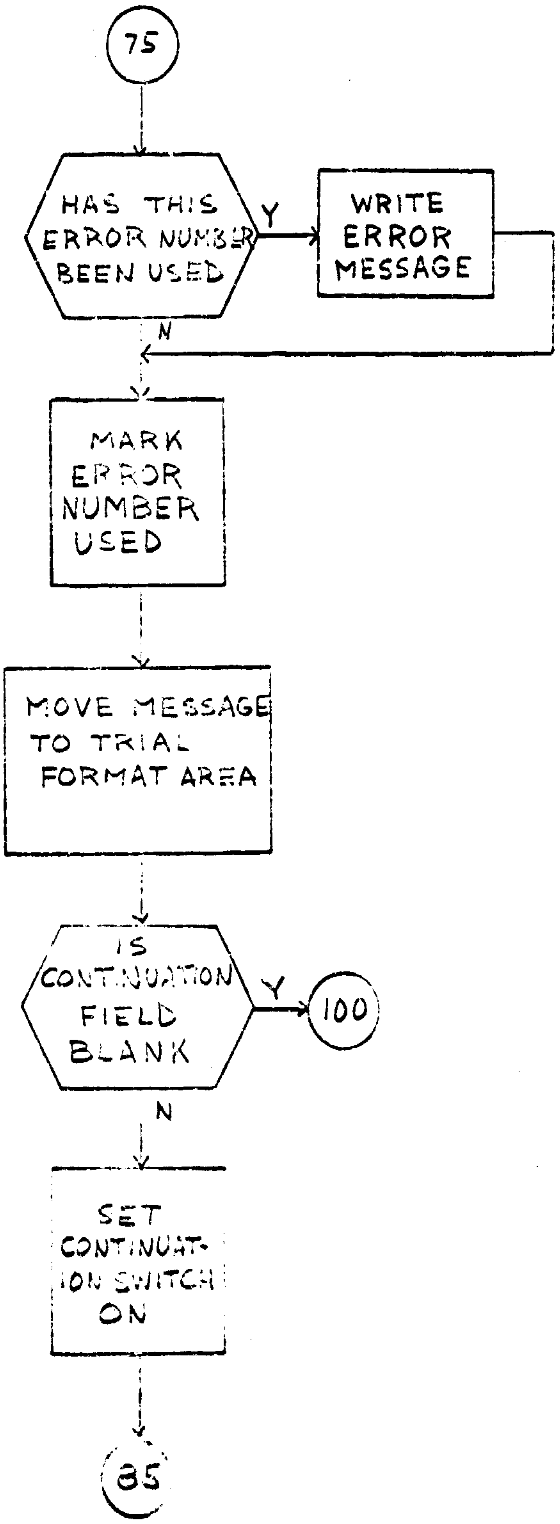

ASSEMBLER
MAIN PROGRAM
CALL
FIRST
PASS
(ASMI)
REWIND
INTERMEDIATE
AND LOCAL
DICTIONARY
FILES
CALL
SECOND
PASS
(ASP)
REWIND
FIRST AND
SECOND PASS
ERROR FILES
CLOSE
CUR
TAPE
FILE
CALL
ERROR
PASS
(ERPASS)
STOP

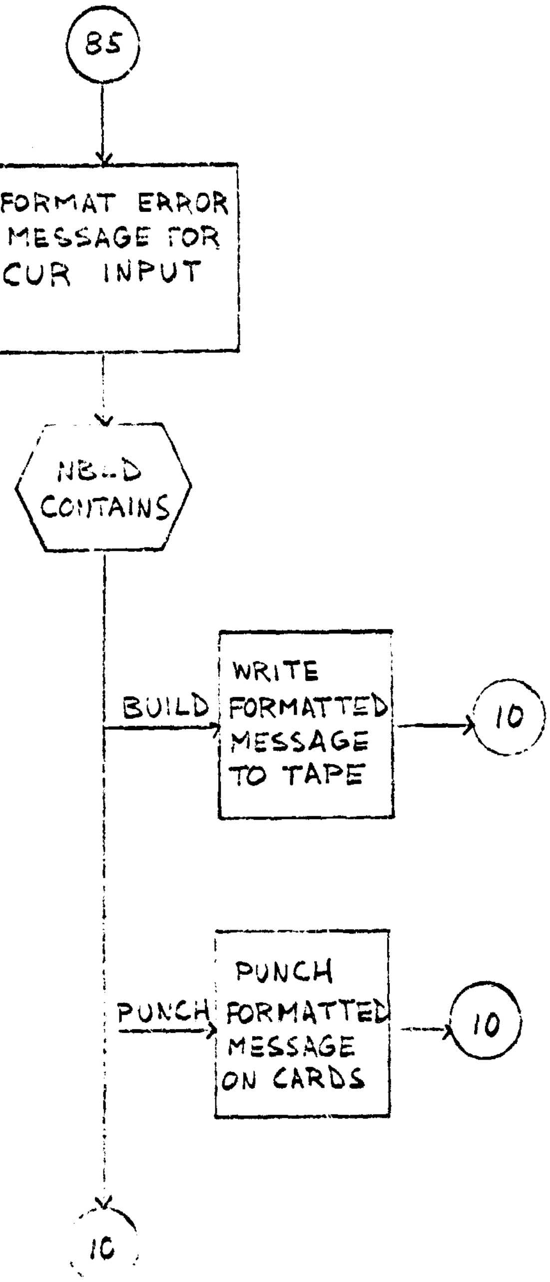
ERROR MESSAGE FORMATTING MAIN PROGRAM

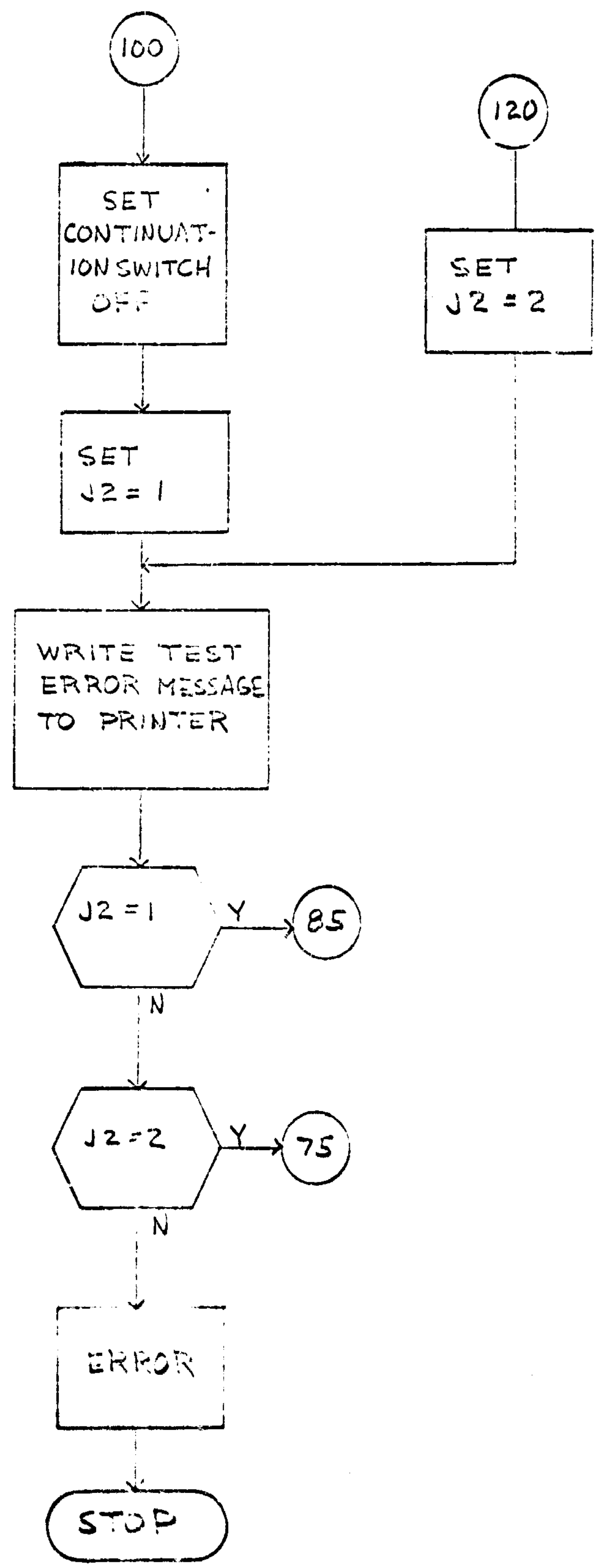
This routine tests and prepares error messages for use by the error pass. A control card is read in containing either the word BUILD, PRINT, or PUNCH. If BUILD is present, then the error messages, after testing, are written out onto a tape in CUR format. If PUNCH is present, then the error messages, after testing, are punched out in form suitable for the CUR program. If PRINT is present, the input error message is only used for testing. Each input message is transformed into a CUR format and is also moved to a FORTRAN format area, from which it is used to write a sample error message. Duplicate error messages, by number, are so indicated.


ERROR MESSAGE FORMATTING PROGRAM
INITIALIZE OUTPUT VARIABLES
READ CONTROL INFORMATION INTO NBLD
SET CONTINUATION SWITCH OFF
SET J3 TO BLANK
NBLD CONTAINS 'BUILD'
N
10
Y
5A


5A
OPEN CUR FILE USING THE NAME 'ERFORM'
10
READ AN ERROR MESSAGE
END OF FILE
Y
N
NBLD CONTAINS BUILD
N
STOP
Y
CLOSE CUR FILE
IS THIS A CONTINUATION CARD
N
75
Y
SET CONTINUATION SWITCH OFF
IS THIS CONTINUATION FIELD > J3
N
120
Y
76

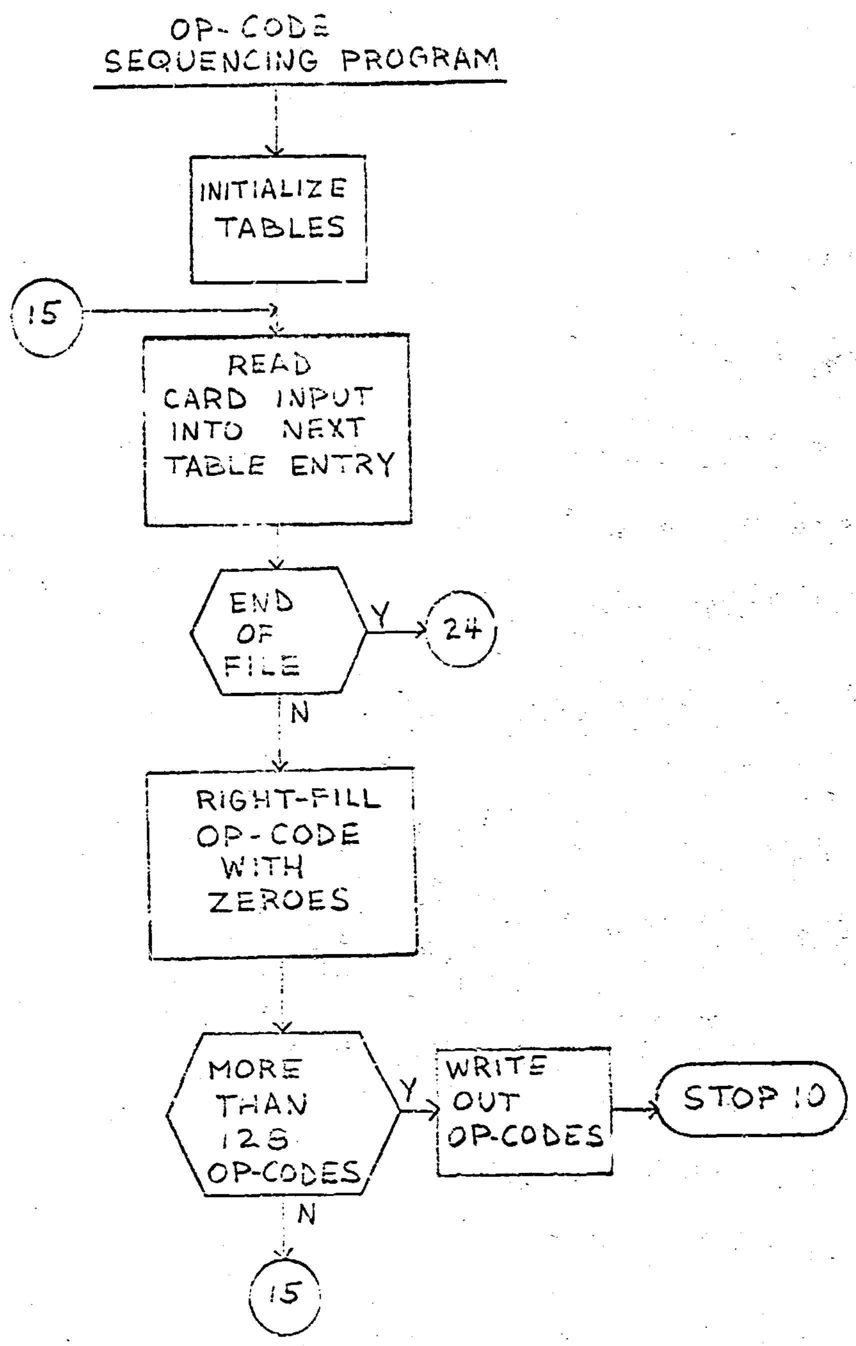

75
HAS THIS ERROR NUMBER BEEN USED
Y
WRITE ERROR MESSAGE
N
MARK ERROR NUMBER USED
MOVE MESSAGE TO TRIAL FORMAT AREA
IS CONTINUATION FIELD BLANK
Y
100
N
SET CONTINUATION SWITCH ON
85

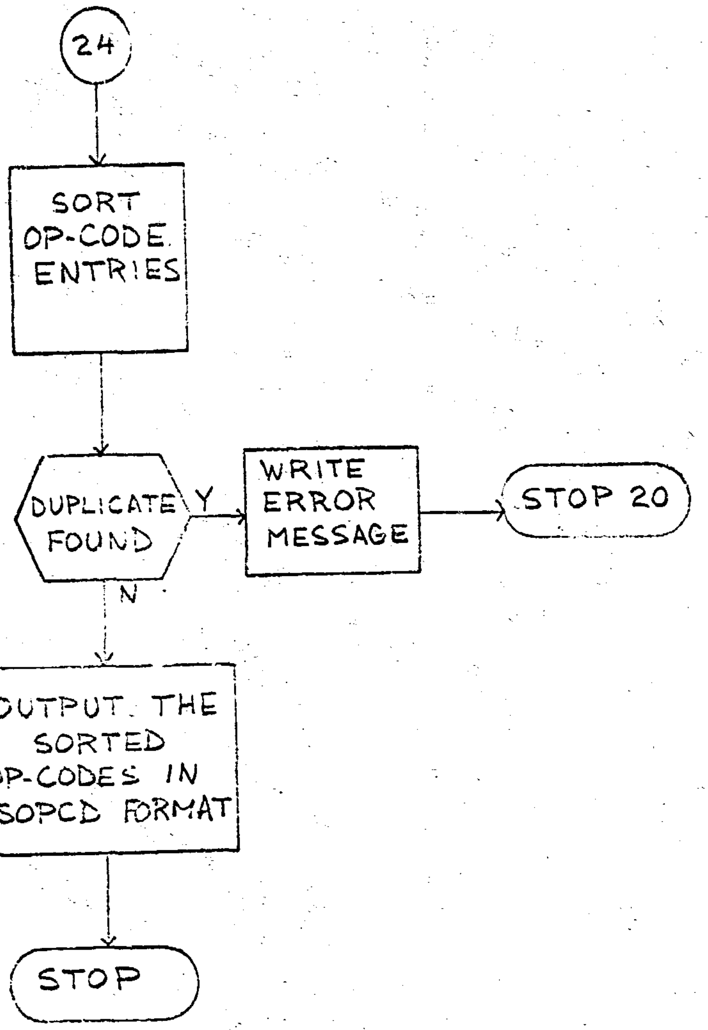

85
FORMAT ERROR MESSAGE FOR CUR INPUT
NBLD CONTAINS
BUILD
WRITE FORMATTED MESSAGE TO TAPE
10
PUNCH
PUNCH FORMATTED MESSAGE ON CARDS
10
10


100
120
SET CONTINUATION SWITCH OFF
SET J2 = 2
SET J2 = 1
WRITE TEST ERROR MESSAGE TO PRINTER
J2 = 1
Y
85
N
J2 = 2
Y
75
N
ERROR
STOP

OP-CODE SEQUENCING PROGRAM

This program reads cards containing op-codes and their associated value. The op-codes can be presented in any order. This program sorts the op-codes and formats them for input to the SOPCD subroutine.


OP-CODE
SEQUENCING PROGRAM
INITIALIZE TABLES
15
READ CARD INPUT INTO NEXT TABLE ENTRY
END OF FILE
Y
24
N
RIGHT-FILL OP-CODE WITH ZEROES
MORE THAN 128 OP-CODES
Y
WRITE OUT OP-CODES
STOP 10
N
15

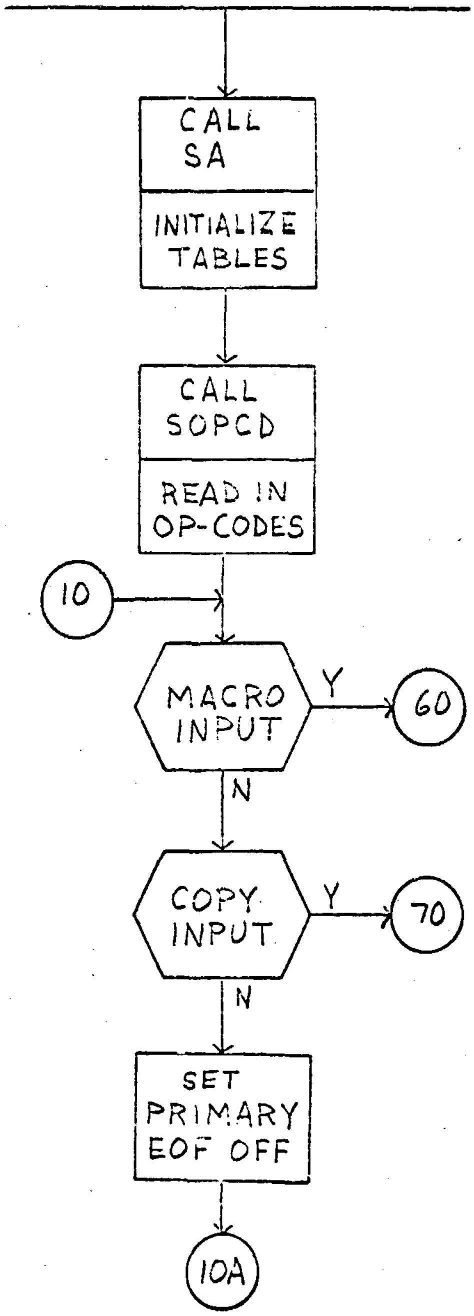

24
SORT OP-CODE ENTRIES
DUPLICATE FOUND
Y
N
WRITE ERROR MESSAGE
STOP 20
OUTPUT THE SORTED OP-CODES IN SOPCD FORMAT
STOP

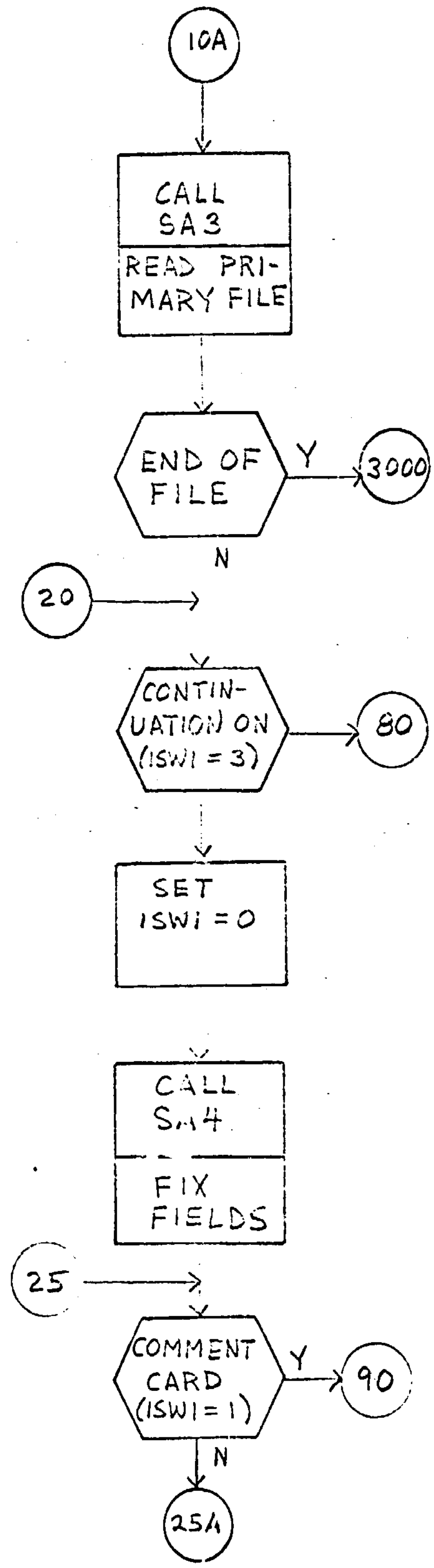
SUBROUTINE ASM1

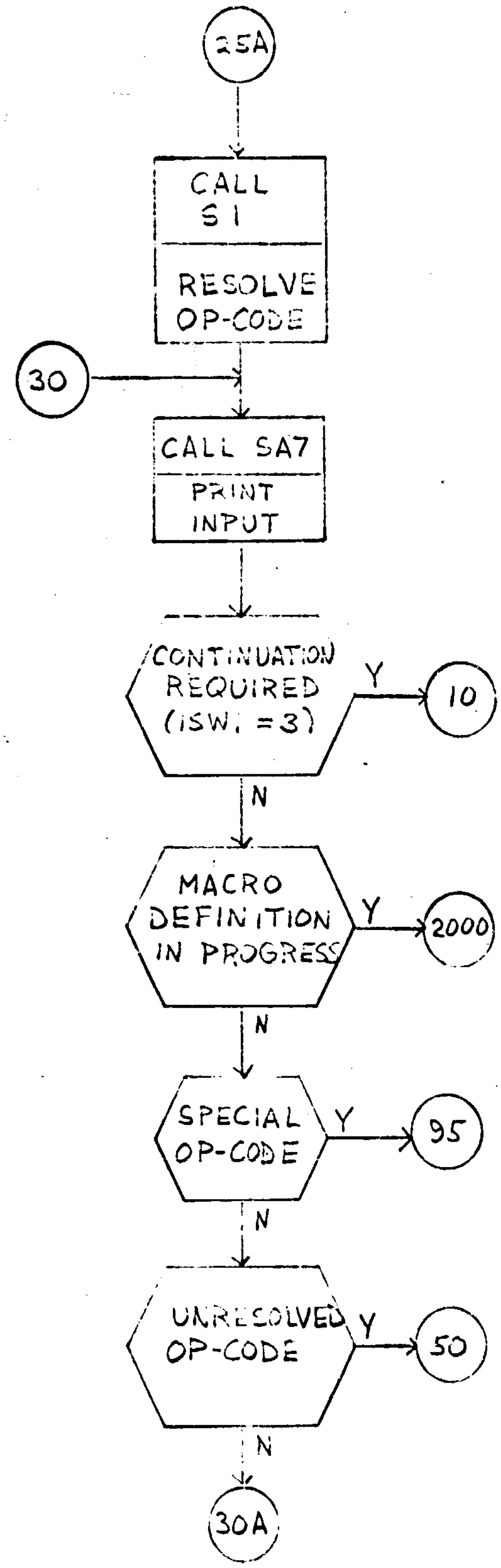
This subroutine makes the first pass through the source statements. It consists of two phases. In the first phase the hardware configuration description statements, and system parameters are read in, processed, and placed in appropriate tables. When all system parameters have been processed, the tables are written out to the simulator input file.

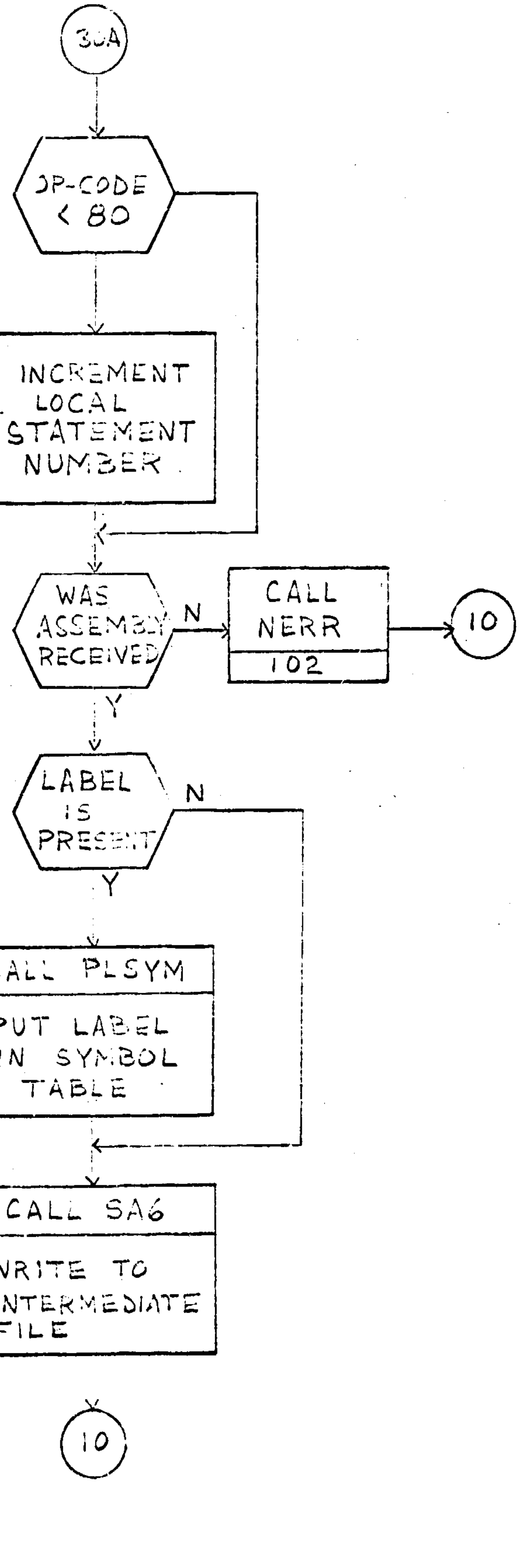
The second phase begins with the receipt of the ASSEMBLY statement. All operating system and worker program representations must follow the ASSEMBLY statement. Furthermore, all macro processing takes place in this phase. In this phase, statement labels and local equates are placed in the local symbol table. A local symbol table is maintained for each job being processed. As each job is ended, this symbol table is written out to a drum storage area. After examining each statement, expanding macro calls as required, the statements are written out to an intermediate file on the drum, from which they will be processed by the second pass.

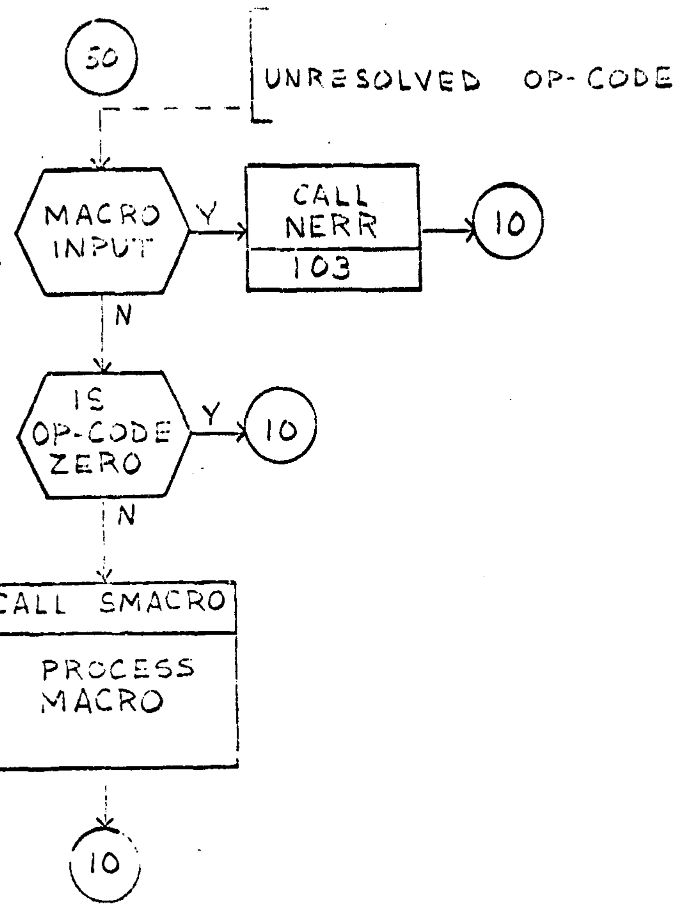
SUBROUTINE ASM1
(FIRST PASS)

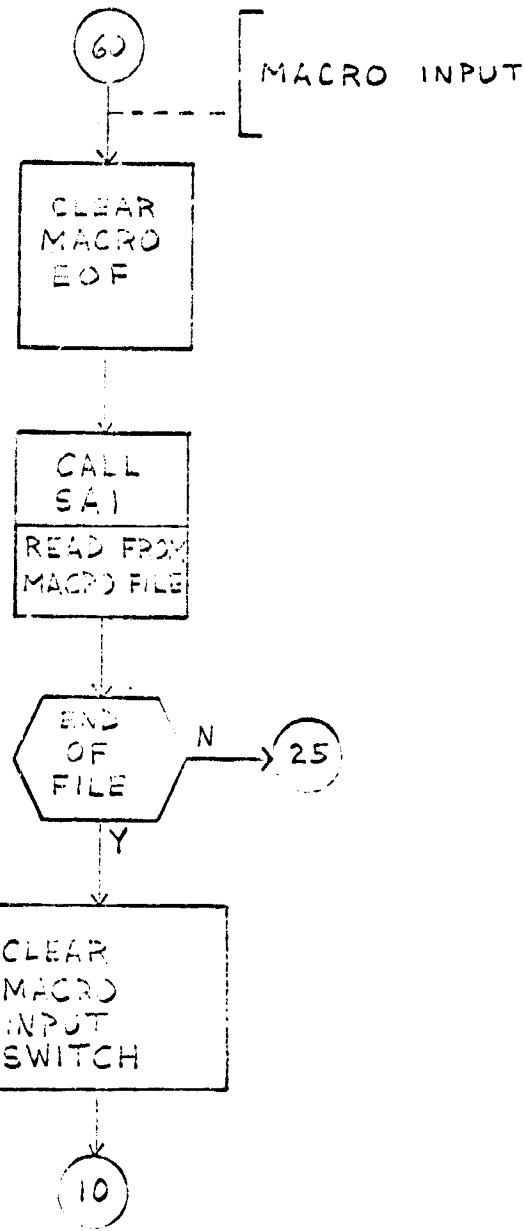


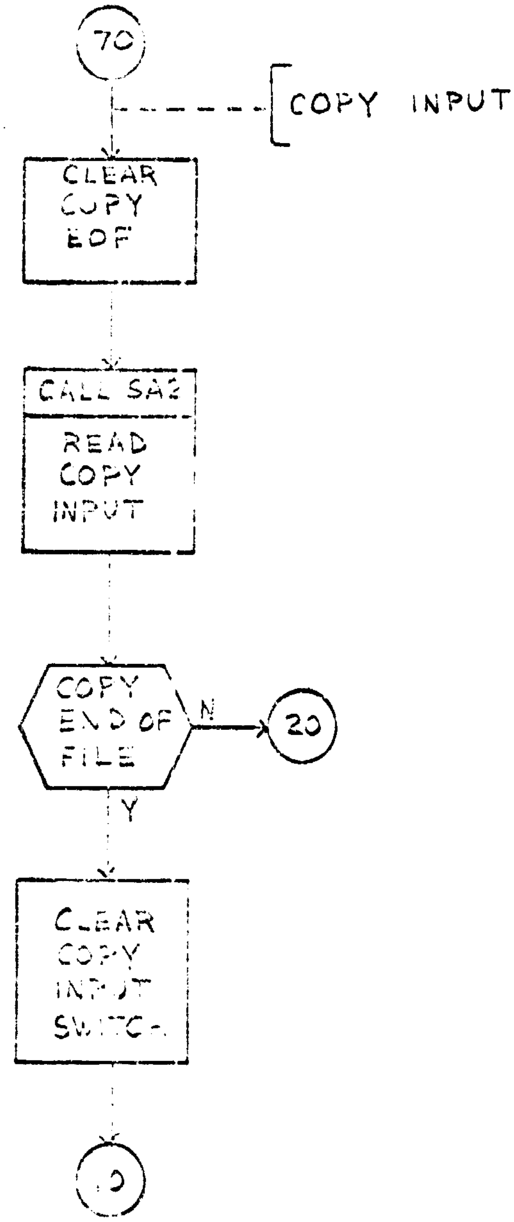

10A
CALL
SA3
READ PRIMARY FILE
END OF
FILE
Y
3000
N
20
CONTINUATION ON
(ISW1 = 3)
80
SET
ISW1 = 0
CALL
SA4
FIX
FIELDS
25
COMMENT
CARD
(ISW1 = 1)
Y
90
N
25A

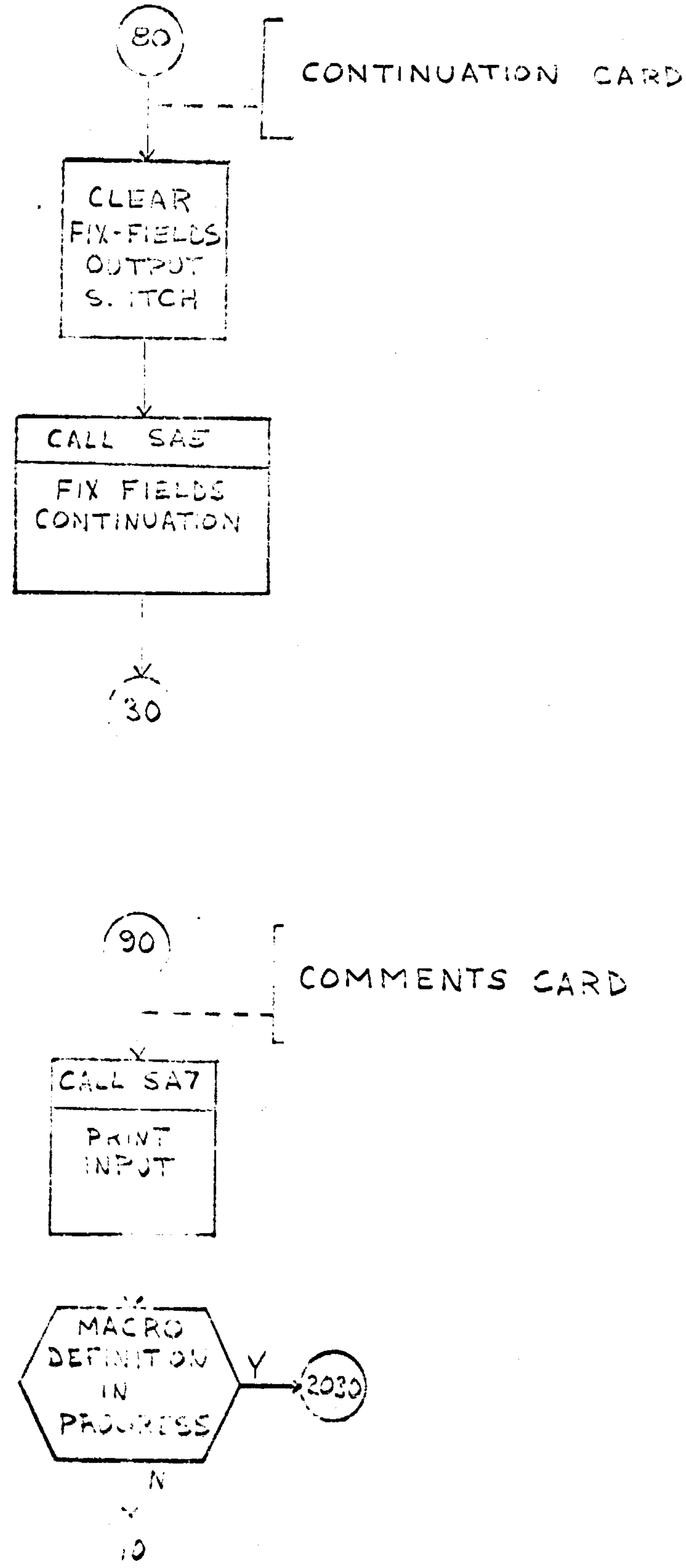

25A
CALL
S1
RESOLVE
OP-CODE
30
CALL SA7
PRINT
INPUT
CONTINUATION
REQUIRED
(ISW1 =3)
Y
10
N
MACRO
DEFINITION
IN PROGRESS
Y
2000
N
SPECIAL
OP-CODE
Y
95
N
UNRESOLVED
OP-CODE
Y
50
N
30A

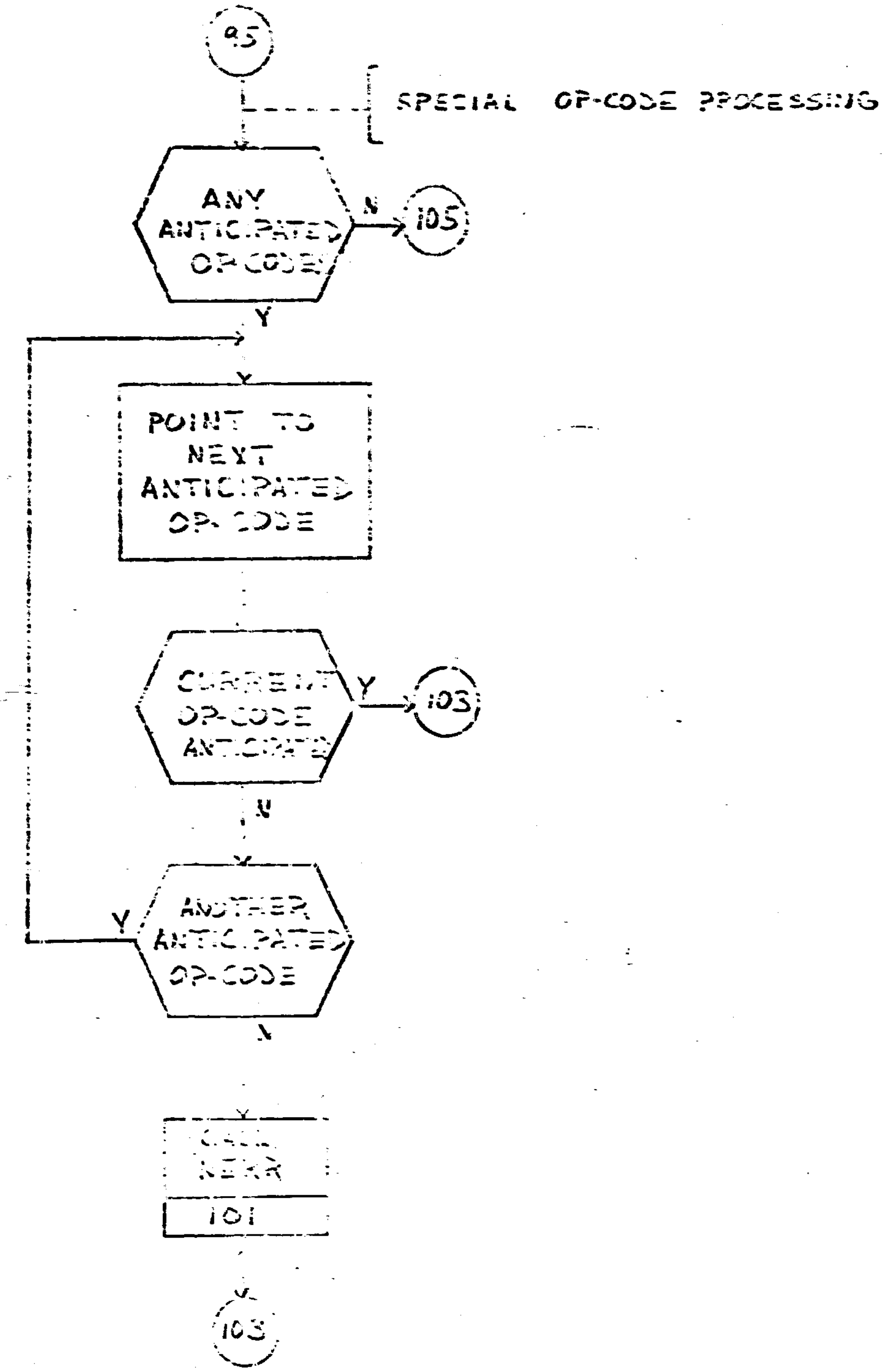

30A
OP-CODE < 80
INCREMENT LOCAL STATEMENT NUMBER
WAS ASSEMBLY RECEIVED
N
CALL NERR
102
10
Y
LABEL IS PRESENT
N
Y
CALL PLSYM
PUT LABEL IN SYMBOL TABLE
CALL SA6
WRITE TO INTERMEDIATE FILE
10

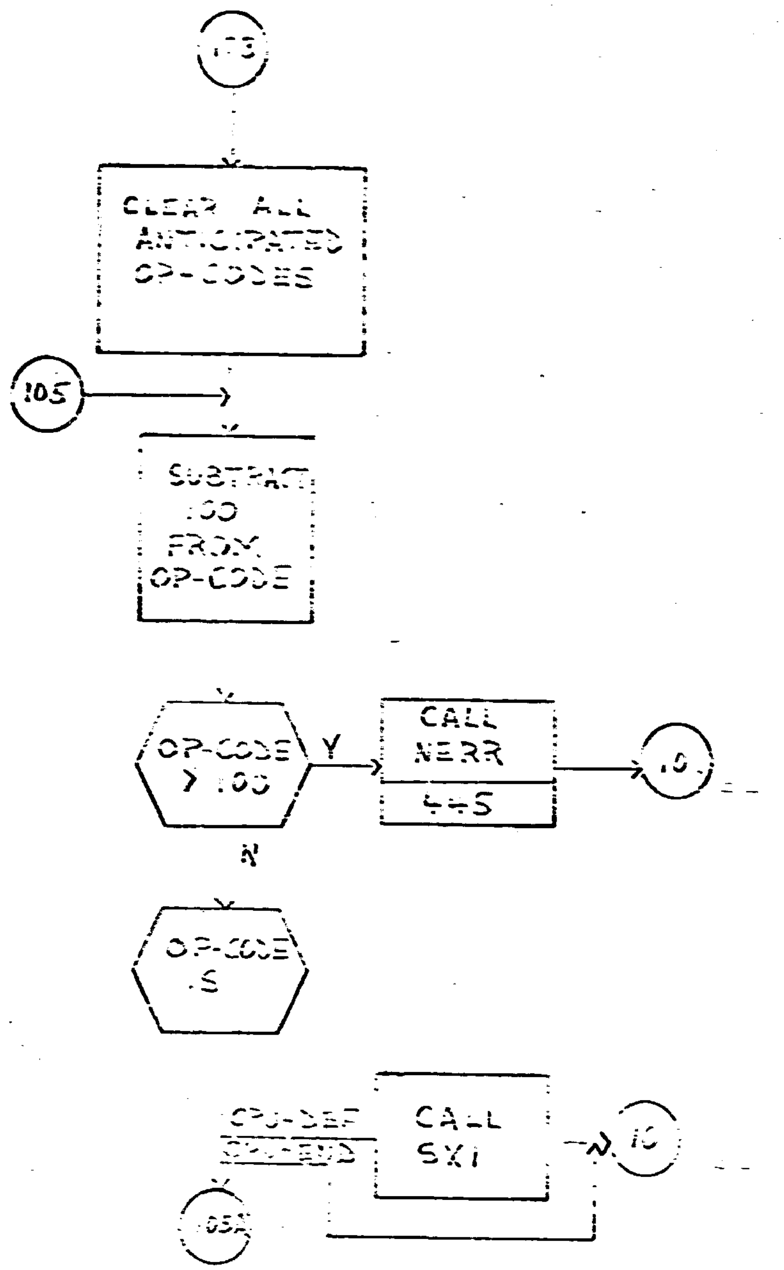

50
UNRESOLVED OP-CODE
MACRO INPUT
Y
CALL NERR
103
10
N
IS OP-CODE ZERO
Y
10
N
CALL SMACRO
PROCESS MACRO
10

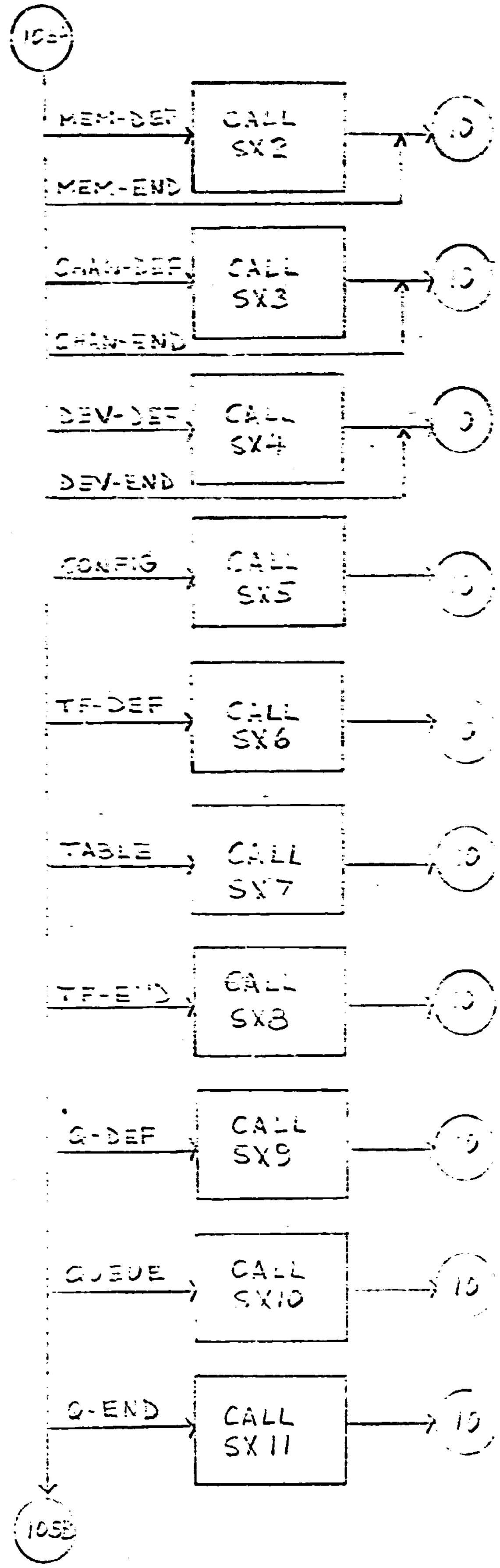

60
MACRO INPUT
CLEAR MACRO EOF
CALL SAI
READ FROM MACRO FILE
END OF FILE
N
25
Y
CLEAR MACRO INPUT SWITCH
10

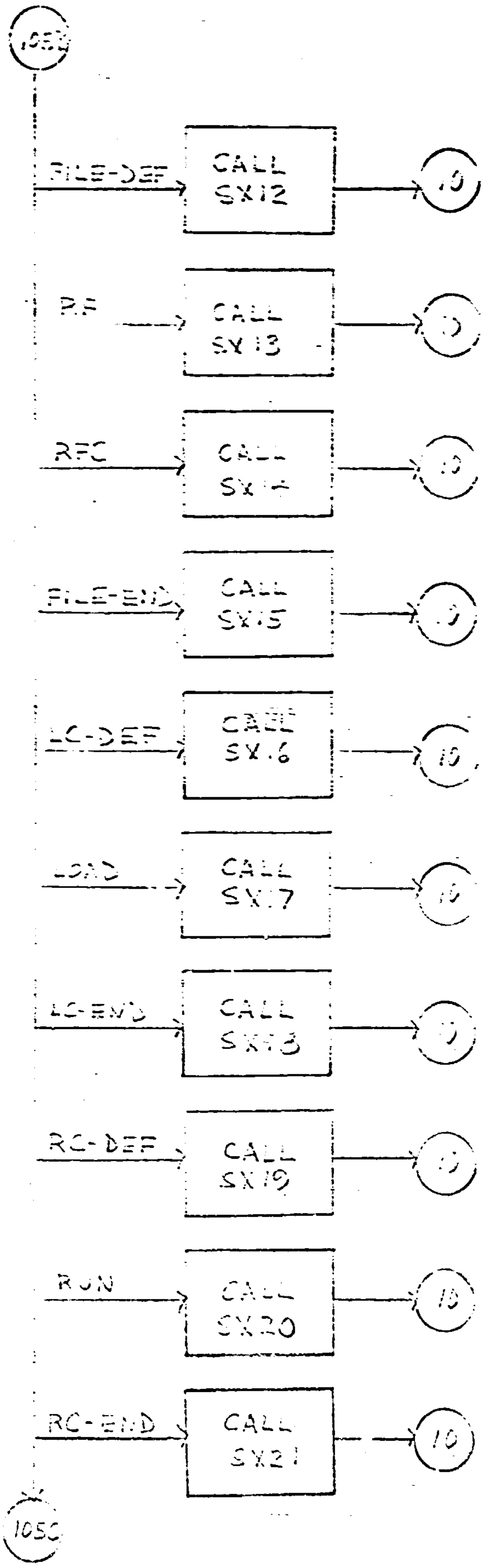

70
COPY INPUT
CLEAR COPY EOF
CALL SA2
READ COPY INPUT
COPY END OF FILE
N
20
Y
CLEAR COPY INPUT SWITCH

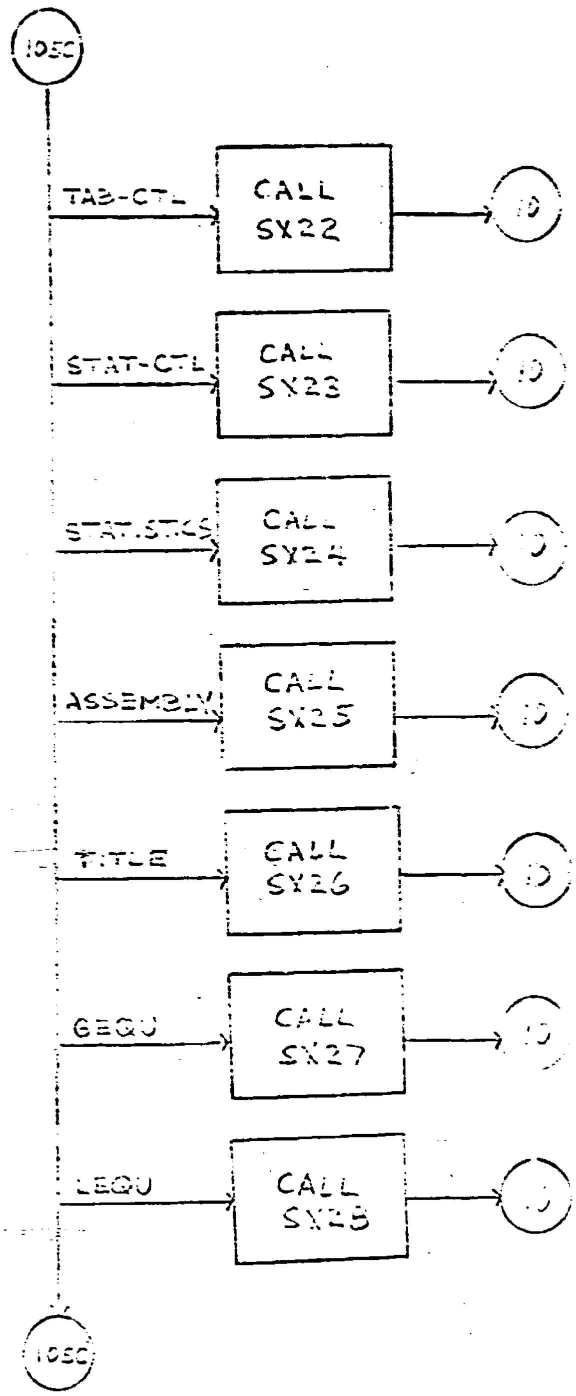

80
CONTINUATION CARD
CLEAR FIX-FIELDS OUTPUT SWITCH
CALL SA5
FIX FIELDS CONTINUATION
30
90
COMMENTS CARD
CALL SA7
PRINT INPUT
MACRO DEFINITION IN PROGRESS
Y
2030
N
10

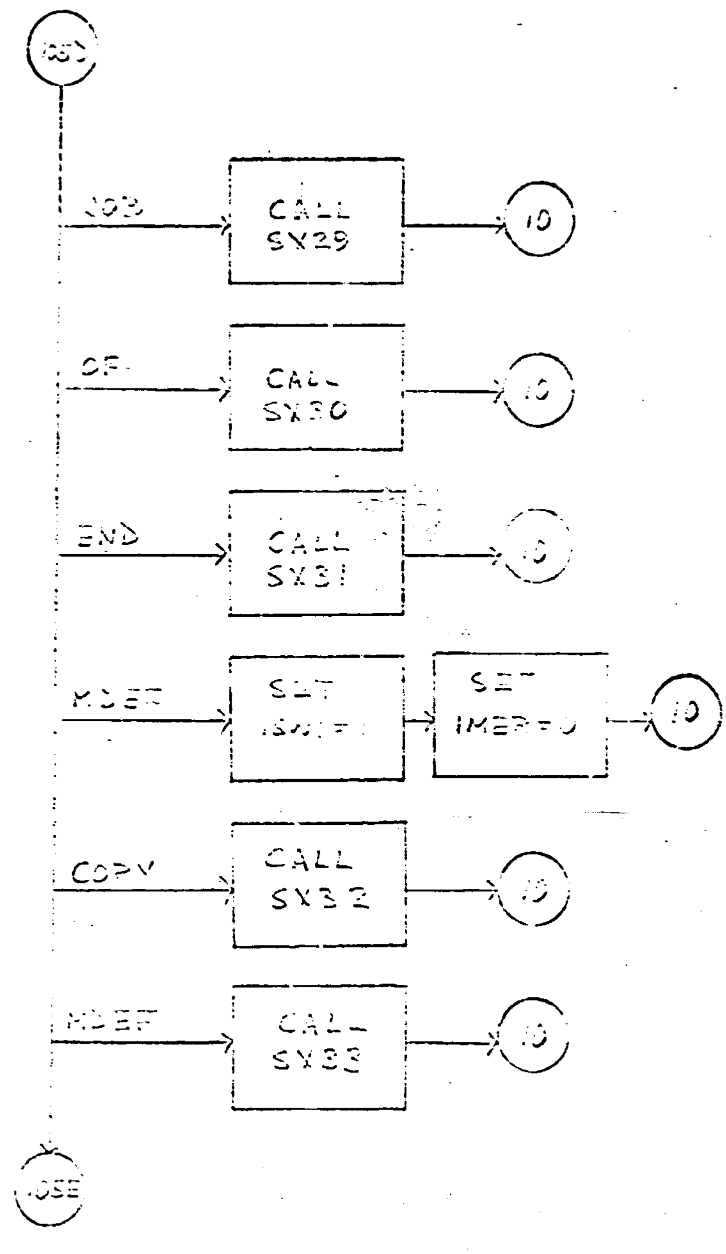

95
SPECIAL OP-CODE PROCESSING
ANY ANTICIPATED OP-CODE
N
105
Y
POINT TO NEXT ANTICIPATED OP-CODE
CURRENT OP-CODE ANTICIPATED
Y
103
N
ANOTHER ANTICIPATED OP-CODE
Y
N
CALL NERR
101
103

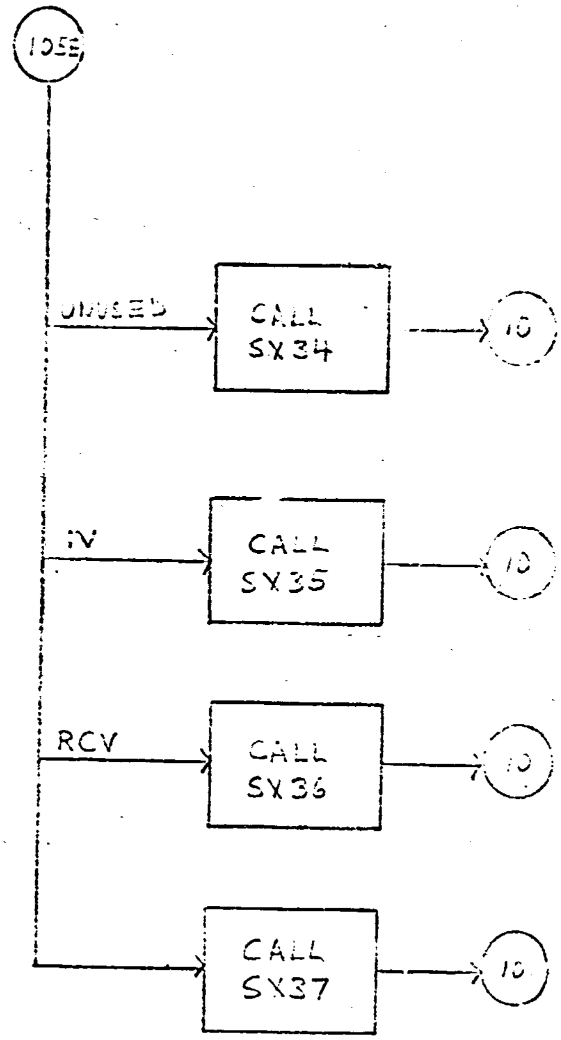

CLEAR ALL
ANTICIPATED
OP-CODES
105
SUBTRACT
100
FROM
OP-CODE
OP-CODE
> 100
Y
CALL
NERR
445
N
CALL
SXI

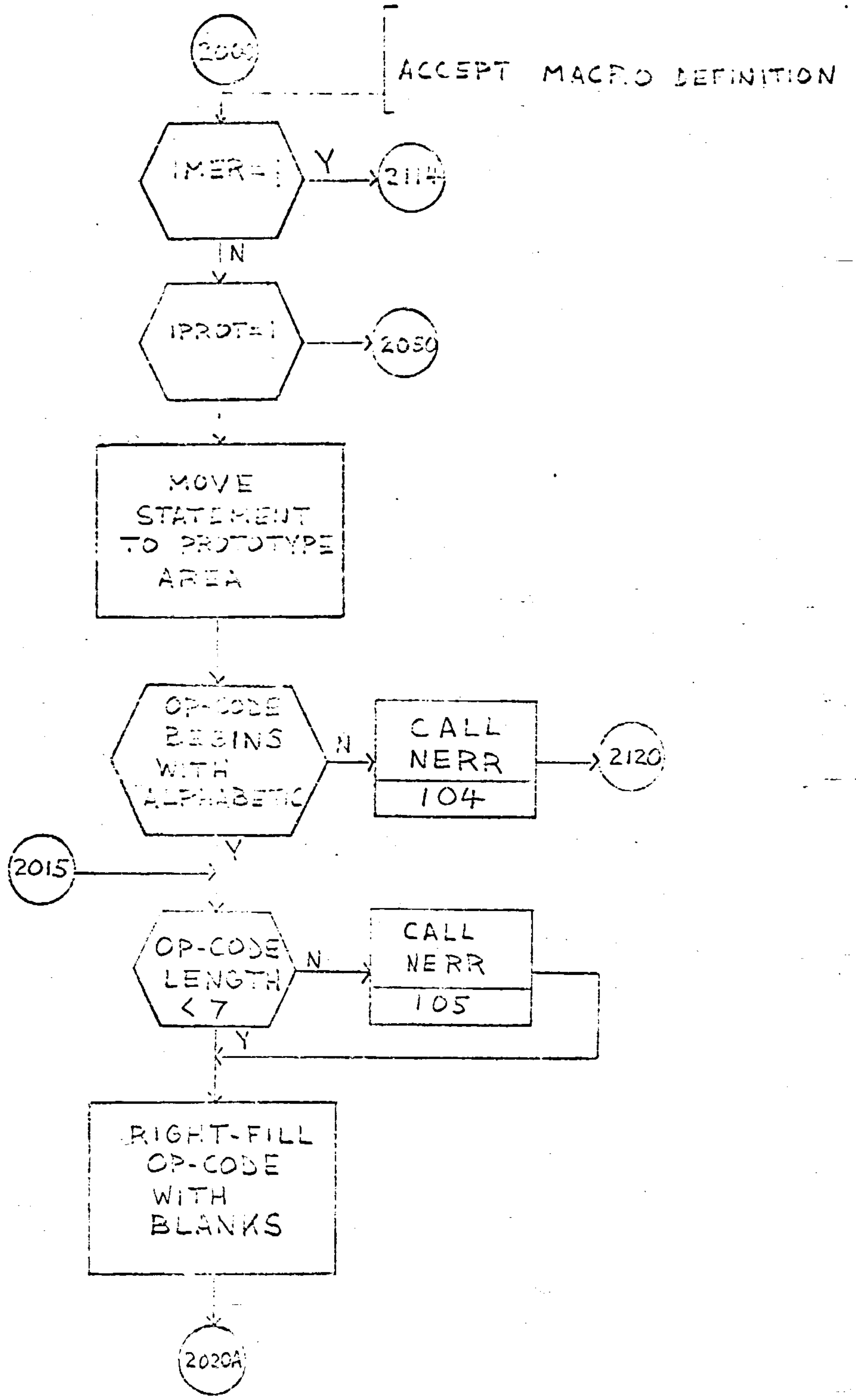

105A
MEM-DEF
MEM-END
CALL SX2
CHAN-DEF
CHAN-END
CALL SX3
DEV-DEF
DEV-END
CALL SX4
CONFIG
CALL SX5
TF-DEF
CALL SX6
TABLE
CALL SX7
TF-END
CALL SX8
Q-DEF
CALL SX9
QUEUE
CALL SX10
Q-END
CALL SX11
10
105B

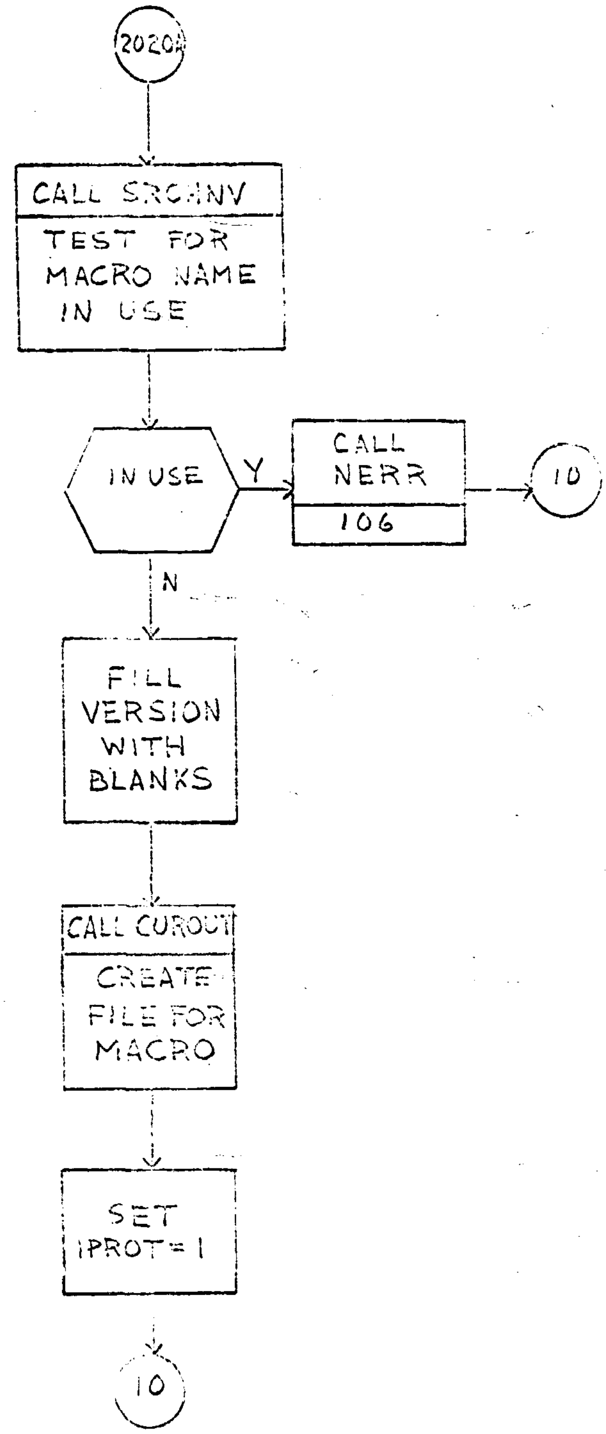

FILE-DEF
CALL SX12
RF
CALL SX13
RFC
CALL SX14
FILE-END
CALL SX15
LC-DEF
CALL SX16
LOAD
CALL SX17
LC-END
CALL SX18
RC-DEF
CALL SX19
RUN
CALL SX20
RC-END
CALL SX21
10
105C

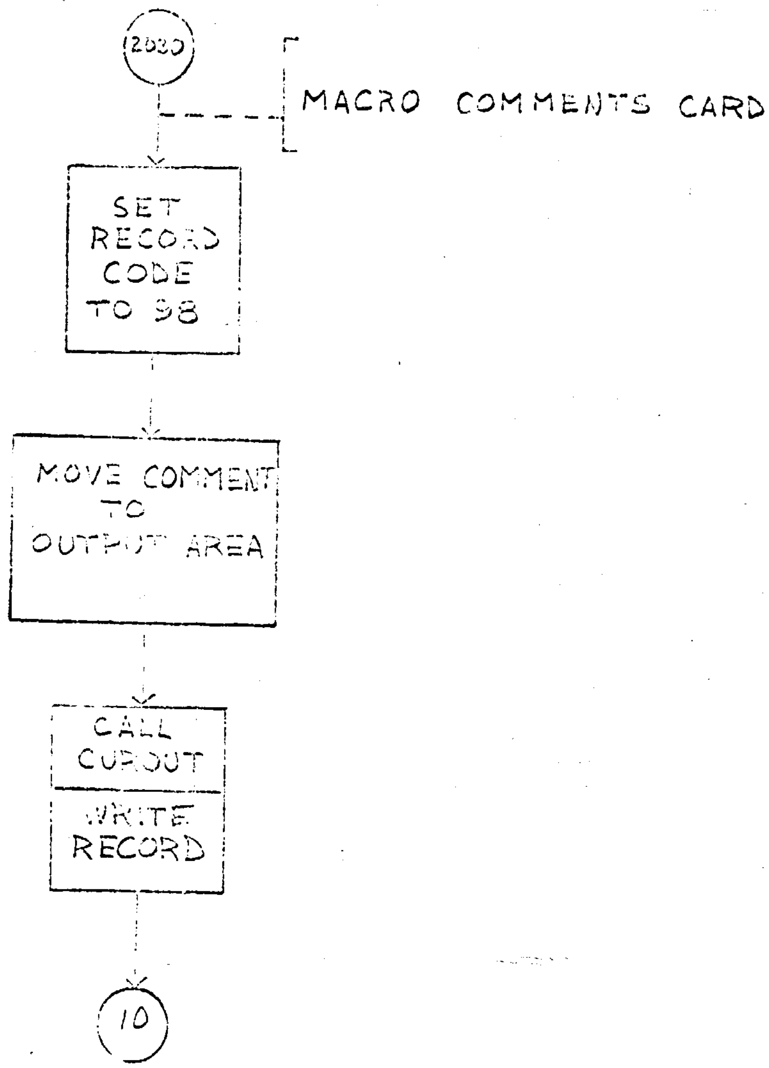

1050
TAB-CTL
CALL SX22
10
STAT-CTL
CALL SX23
10
STATISTICS
CALL SX24
10
ASSEMBLY
CALL SX25
10
TITLE
CALL SX26
10
GEQU
CALL SX27
10
LEQU
CALL SX28
10
1050

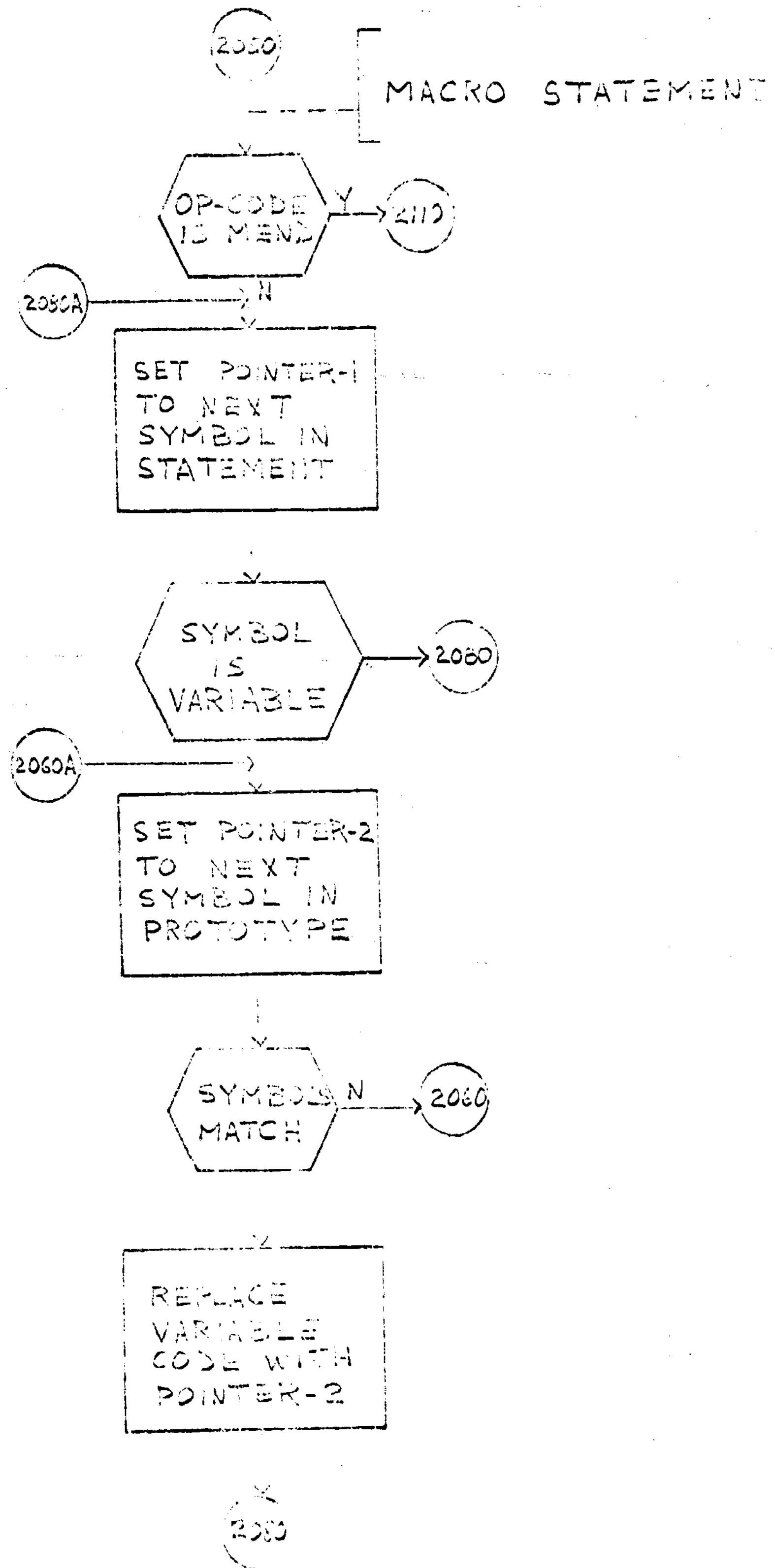

JOB
CALL SX29
10
OP
CALL SX30
10
END
CALL SX31
10
MDEF
SET
SET
10
COPY
CALL SX32
10
MDEF
CALL SX33
10

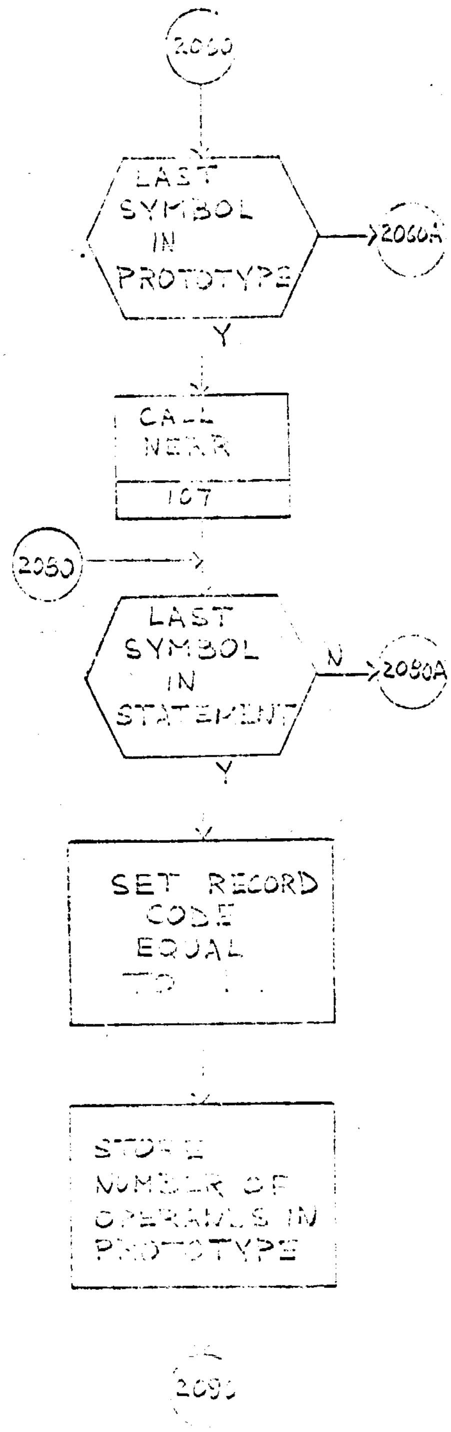

105E
UNUSED
CALL SX34
10
TV
CALL SX35
10
RCV
CALL SX36
10
CALL SX37
10

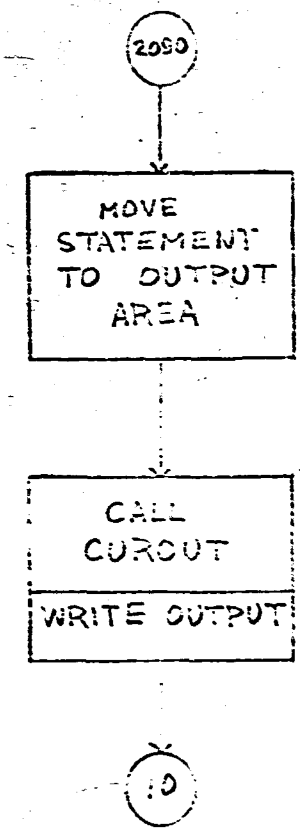

2000
ACCEPT MACRO DEFINITION
IMER=1
Y
2114
N
IPROT=1
2050
MOVE STATEMENT TO PROTOTYPE AREA
OP-CODE BEGINS WITH ALPHABETIC
N
CALL NERR
104
2120
Y
2015
OP-CODE LENGTH < 7
N
CALL NERR
105
Y
RIGHT-FILL OP-CODE WITH BLANKS
2020A

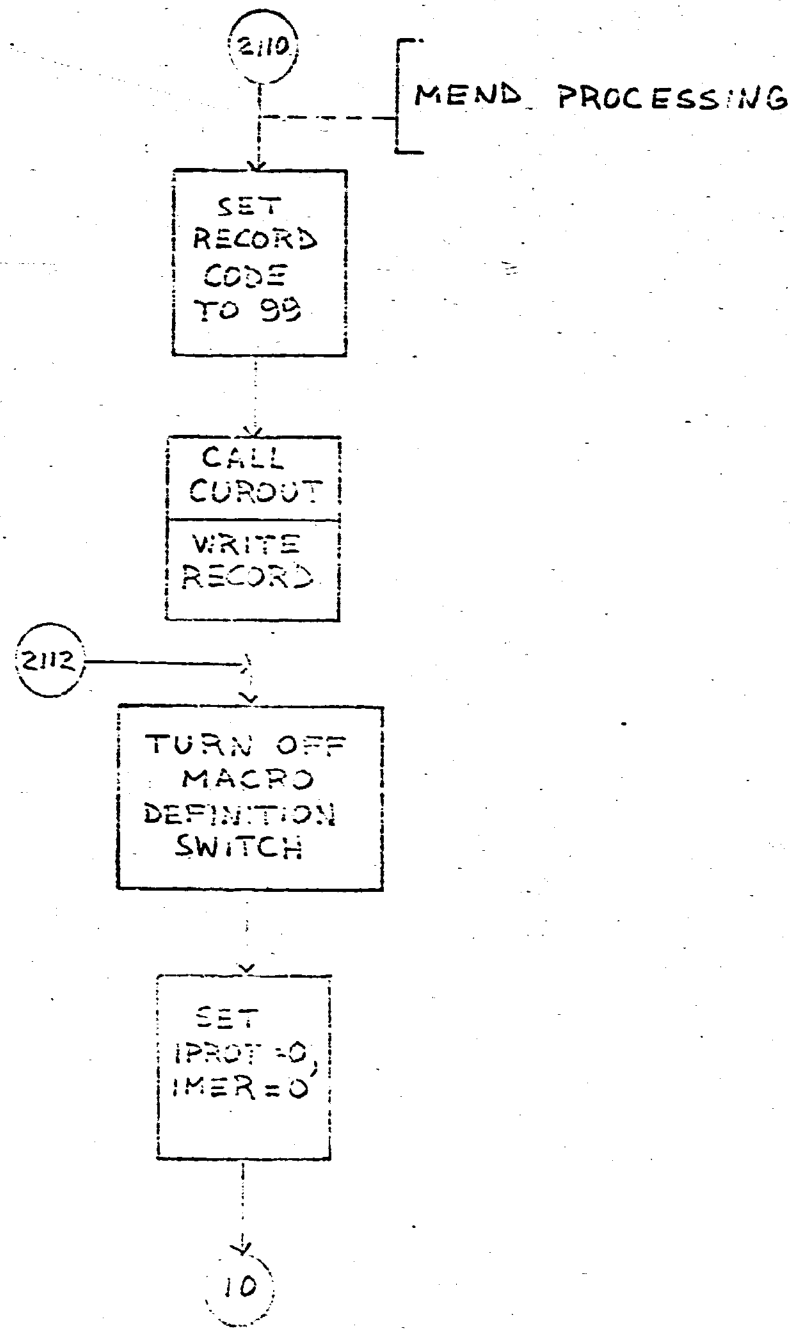

2020A
CALL SRCHNV
TEST FOR MACRO NAME IN USE
IN USE
Y
CALL NERR
106
10
N
FILL VERSION WITH BLANKS
CALL CUROUT
CREATE FILE FOR MACRO
SET IPROT=1
10

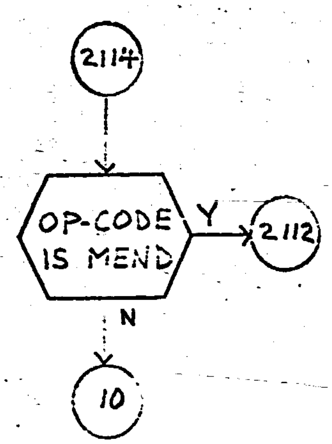

2020
MACRO COMMENTS CARD
SET RECORD CODE TO 98
MOVE COMMENT TO OUTPUT AREA
CALL CUROUT
WRITE RECORD
10

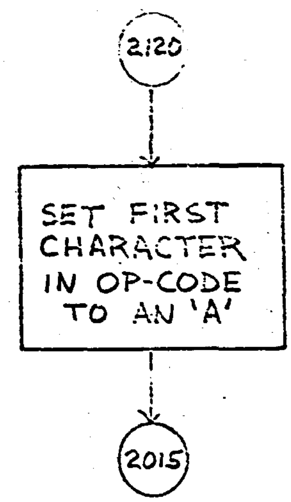

2080
MACRO STATEMENT
OP-CODE IS MEND
Y
2110
N
2080A
SET POINTER-1 TO NEXT SYMBOL IN STATEMENT
SYMBOL IS VARIABLE
2080
2060A
SET POINTER-2 TO NEXT SYMBOL IN PROTOTYPE
SYMBOLS MATCH
N
2060
REPLACE VARIABLE CODE WITH POINTER-2
2080

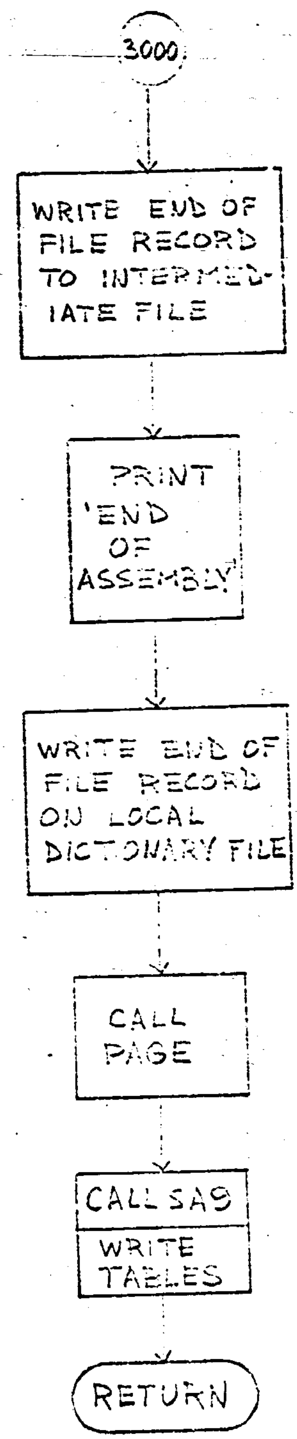

2060
LAST SYMBOL IN PROTOTYPE
2060A
Y
CALL NERR
107
2080
LAST SYMBOL IN STATEMENT
N
2080A
Y
SET RECORD CODE EQUAL TO 1
STORE NUMBER OF OPERANDS IN PROTOTYPE
2090


2090
MOVE STATEMENT TO OUTPUT AREA
CALL CUROUT
WRITE OUTPUT
10


2110
MEND PROCESSING
SET RECORD CODE TO 99
CALL CUROUT
WRITE RECORD
2112
TURN OFF MACRO DEFINITION SWITCH
SET IPROT = 0, IMER = 0
10


2114
OP-CODE IS MEND
Y
2112
N
10

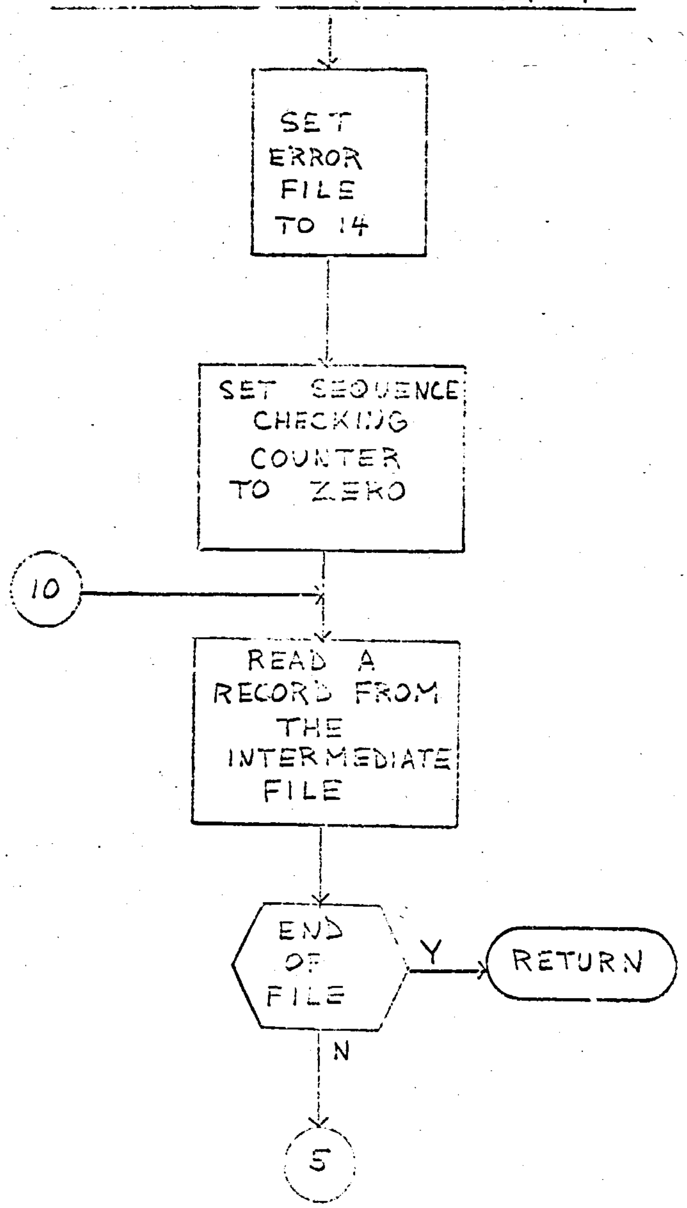

2120
SET FIRST CHARACTER IN OP-CODE TO AN 'A'
2015

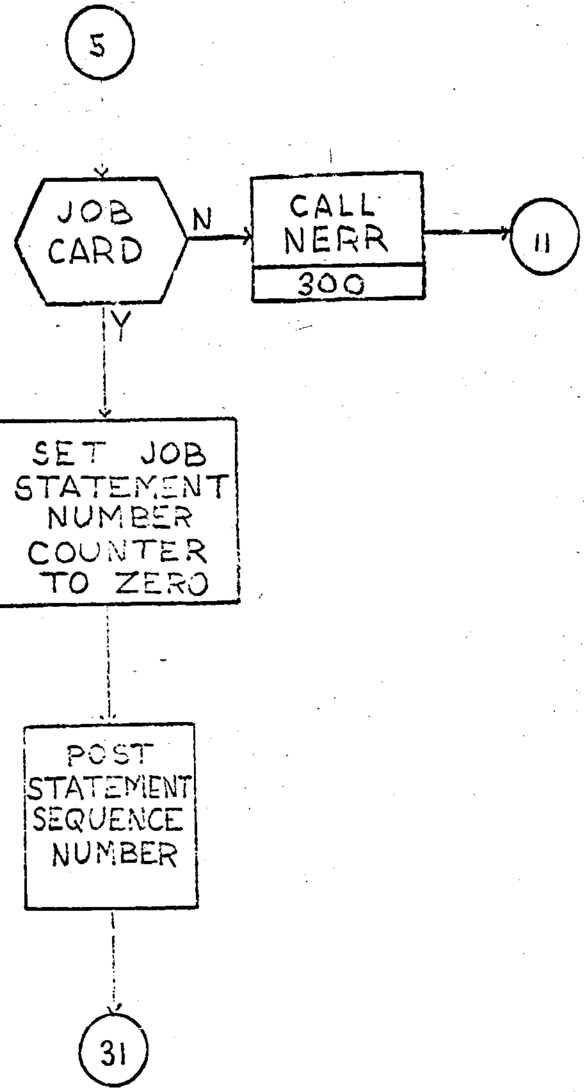

3000
WRITE END OF FILE RECORD TO INTERMED-IATE FILE
PRINT 'END OF ASSEMBLY'
WRITE END OF FILE RECORD ON LOCAL DICTIONARY FILE
CALL PAGE
CALL SA9
WRITE TABLES
RETURN

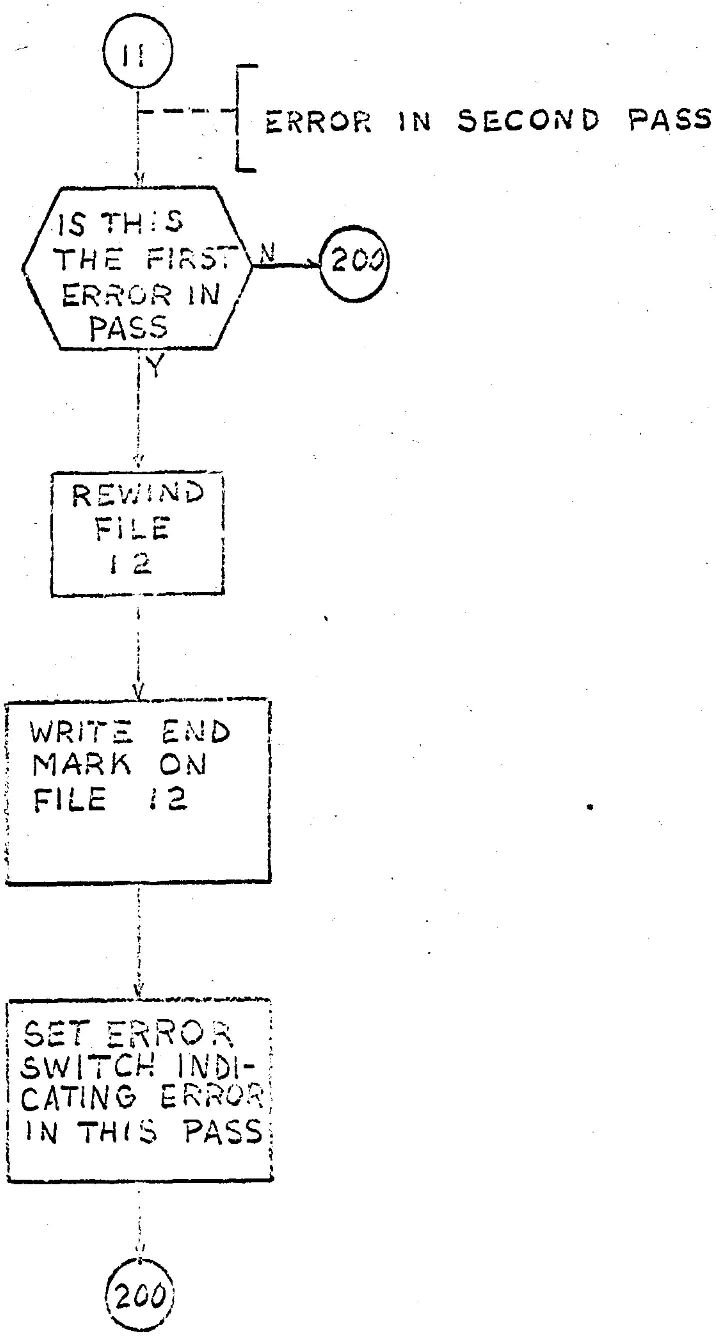
SUBROUTINE ASP

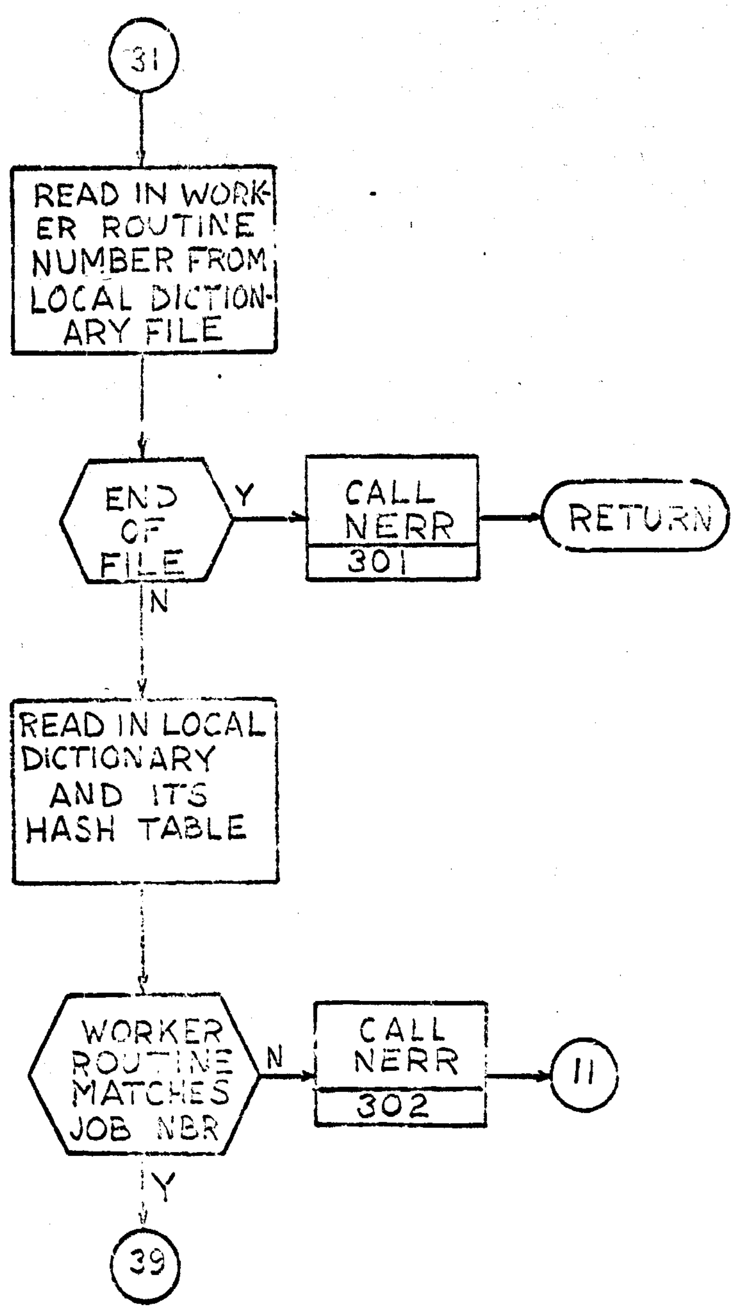
This is the assembler second pass routine. It processes the intermediate file records into assembler input records. It is a table driven translator. For each statement, there is a table entry. In the entry there is a field to indicate whether pre-operand processing is required, whether post operand processing is required, operand processing for each of the six possible operands for the given statement, and a field containing the length of the statement as it is used by the simulator. As each statement is read from the intermediate file the table entry is consulted and the appropriate pre, post, and operand processing routines are used.

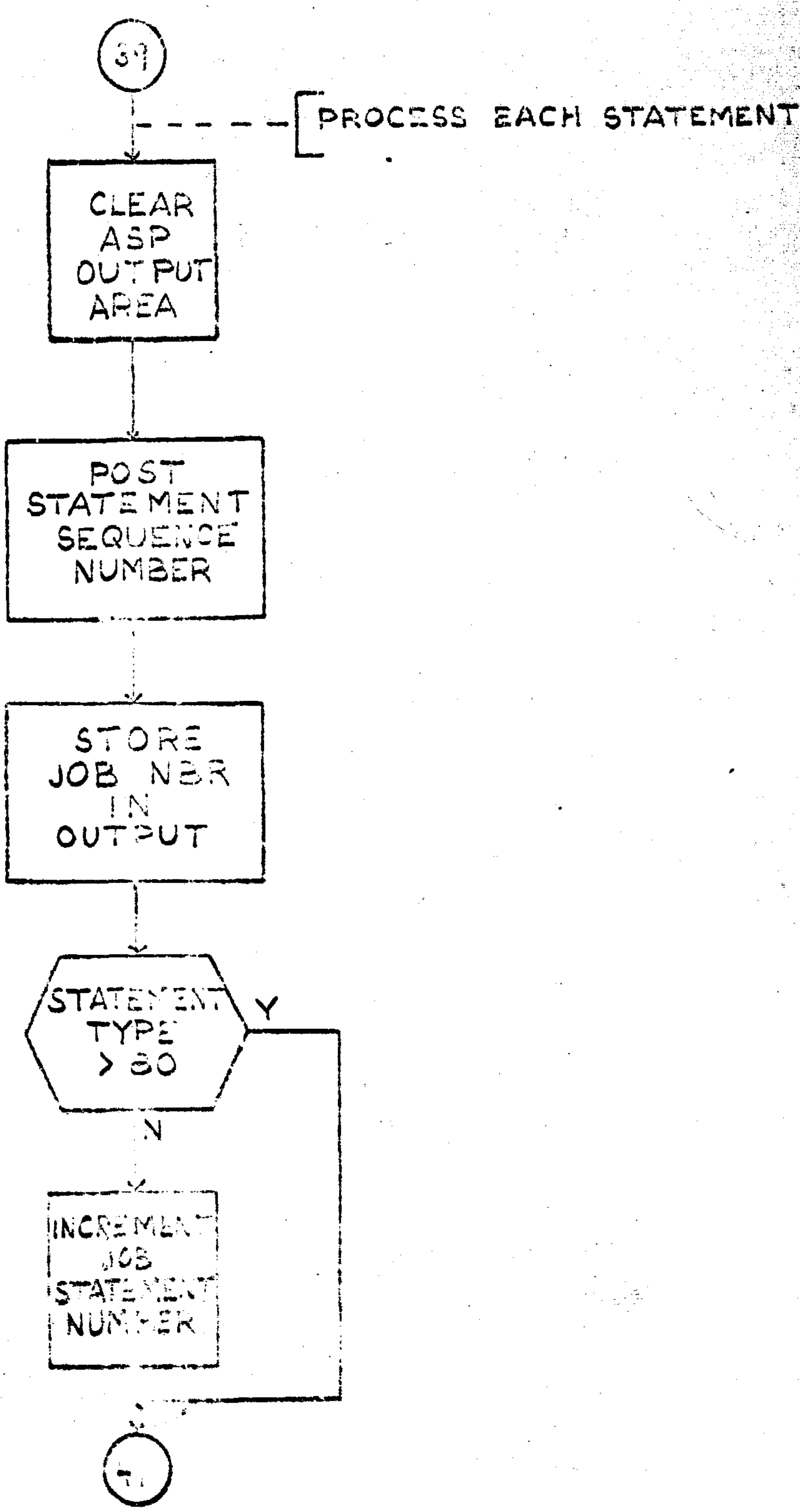
SUBROUTINE ASP
(ASSEMBLER SECOND PASS)

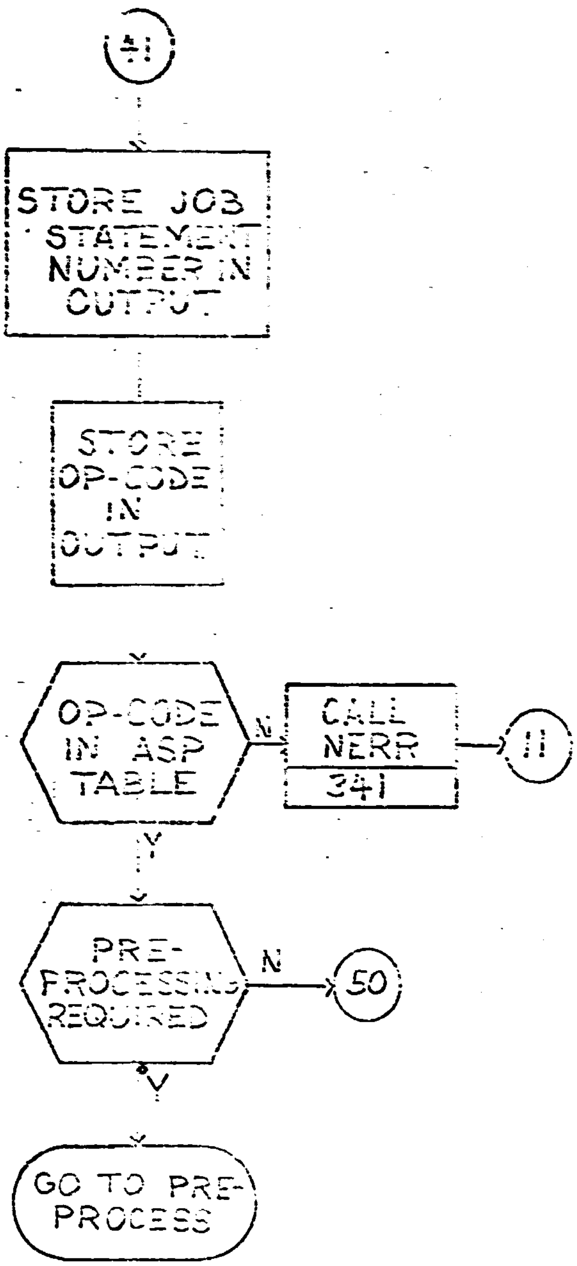


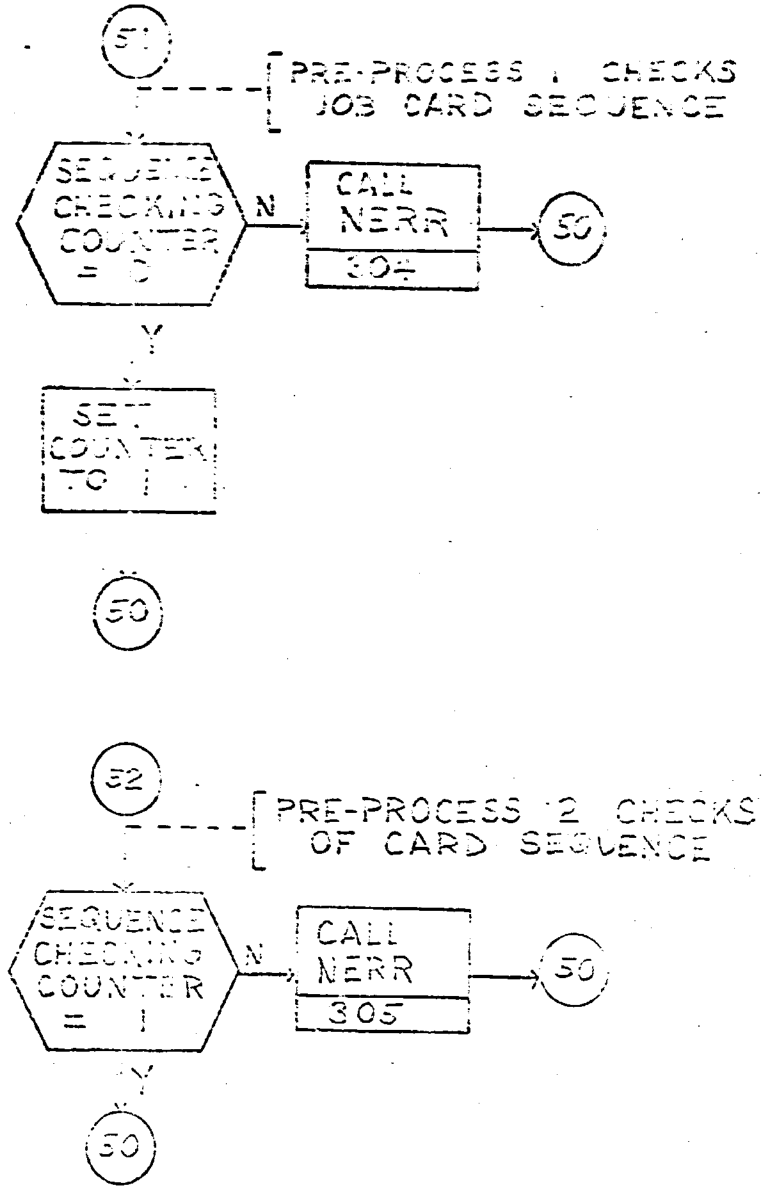

5
JOB CARD
N
CALL NERR
300
11
Y
SET JOB STATEMENT NUMBER COUNTER TO ZERO
POST STATEMENT SEQUENCE NUMBER
31

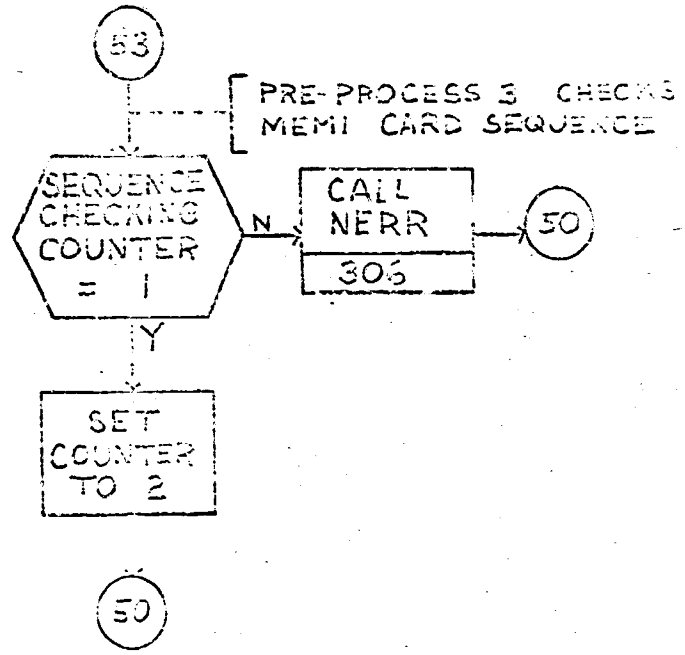

11
ERROR IN SECOND PASS
IS THIS THE FIRST ERROR IN PASS
N
200
Y
REWIND FILE 12
WRITE END MARK ON FILE 12
SET ERROR SWITCH INDICATING ERROR IN THIS PASS
200

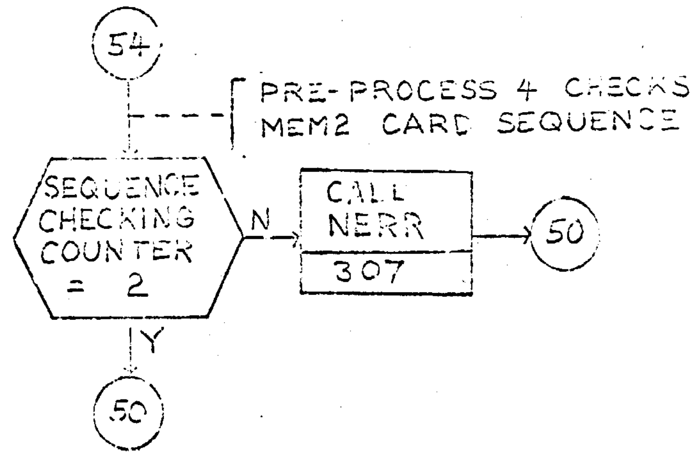

31
READ IN WORKER ROUTINE NUMBER FROM LOCAL DICTIONARY FILE
END OF FILE
Y
CALL NERR
301
RETURN
N
READ IN LOCAL DICTIONARY AND ITS HASH TABLE
WORKER ROUTINE MATCHES JOB NBR
N
CALL NERR
302
11
Y
39

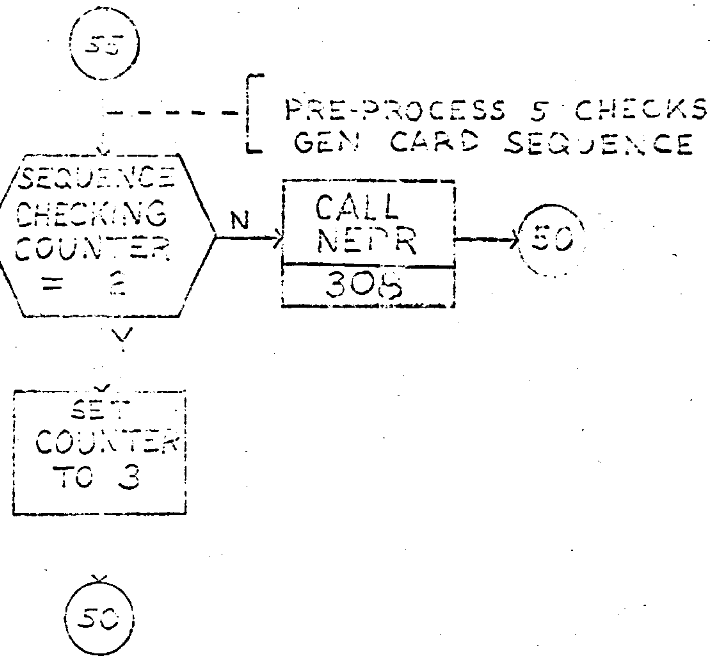

39
PROCESS EACH STATEMENT
CLEAR ASP OUTPUT AREA
POST STATEMENT SEQUENCE NUMBER
STORE JOB NBR IN OUTPUT
STATEMENT TYPE > 80
Y
N
INCREMENT JOB STATEMENT NUMBER


41
STORE JOB STATEMENT NUMBER IN OUTPUT
STORE OP-CODE IN OUTPUT
OP-CODE IN ASP TABLE
N
CALL NERR
341
11
Y
PRE-PROCESSING REQUIRED
N
50
Y
GO TO PRE-PROCESS

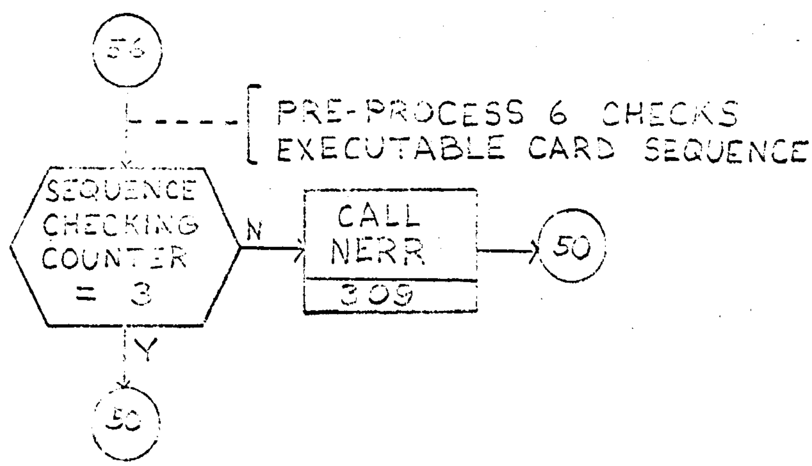

51
PRE-PROCESS 1 CHECKS JOB CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 0
N
CALL NERR
304
50
Y
SET COUNTER TO 1
50
52
PRE-PROCESS 2 CHECKS OF CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 1
N
CALL NERR
305
50
Y
50


53
PRE-PROCESS 3 CHECKS
MEM1 CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 1
N
CALL NERR
306
50
Y
SET COUNTER TO 2
50

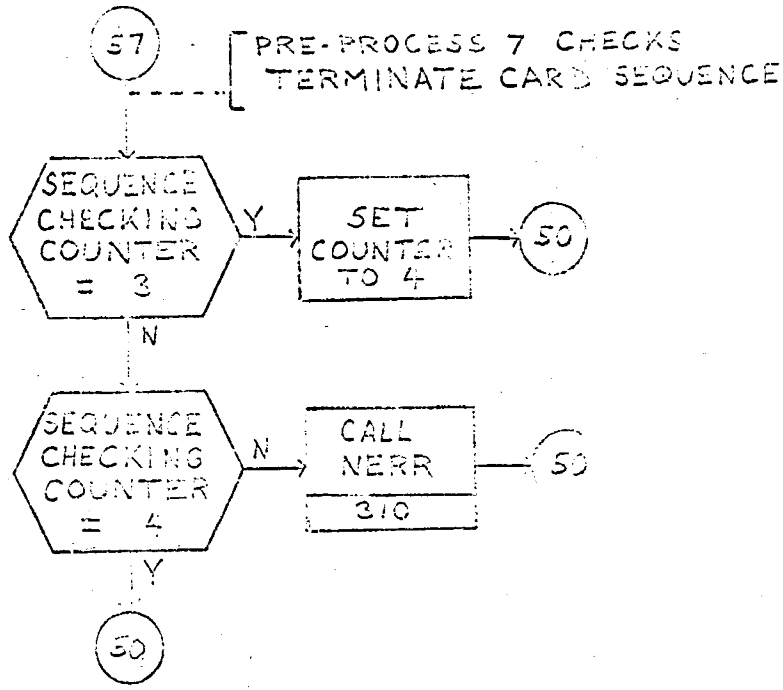

54
PRE-PROCESS 4 CHECKS
MEM2 CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 2
N
CALL NERR
307
50
Y
50

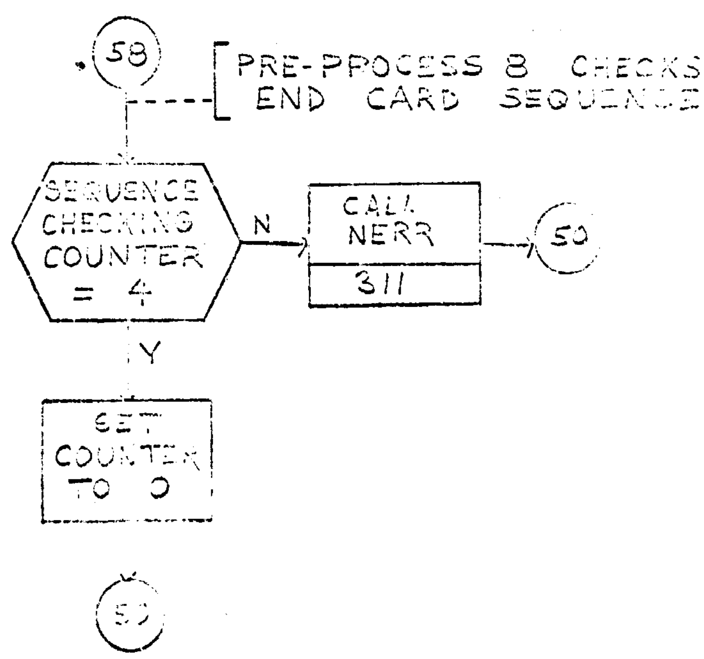


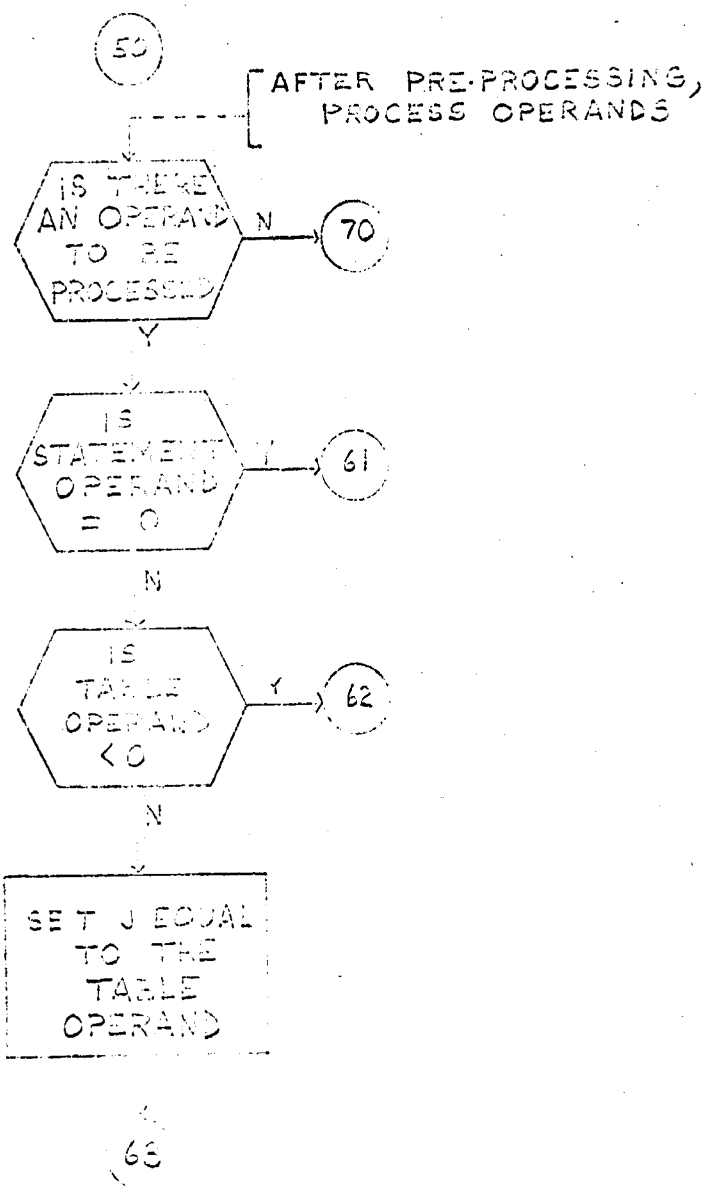
55
PRE-PROCESS 5 CHECKS
GEN CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 2
N
Y
CALL NERR
308
50
SET COUNTER TO 3
50



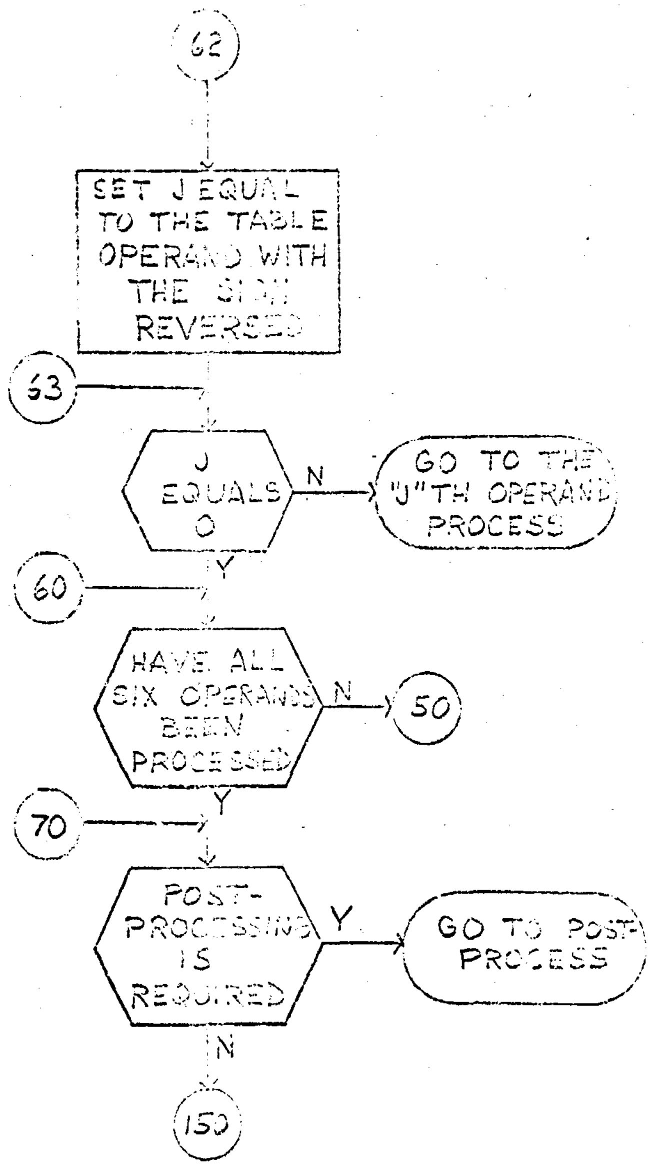
56
PRE-PROCESS 6 CHECKS
EXECUTABLE CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 3
N
Y
CALL NERR
309
50
50

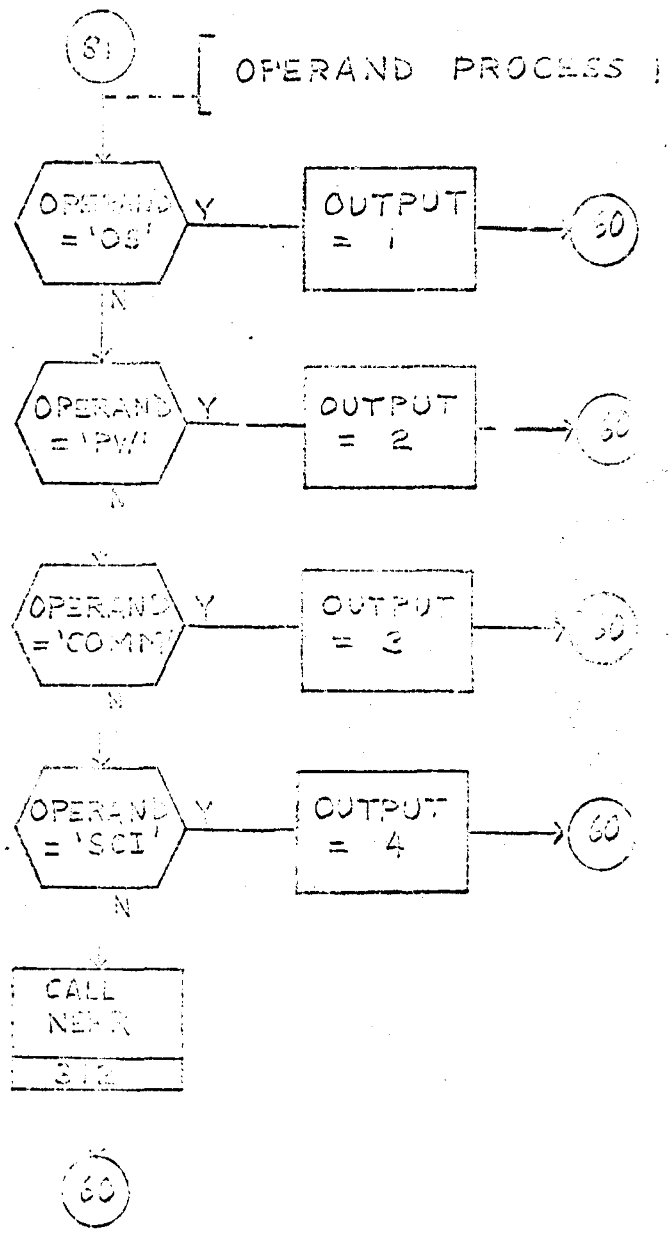

57
PRE-PROCESS 7 CHECKS
TERMINATE CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 3
Y
SET COUNTER TO 4
50
N
SEQUENCE CHECKING COUNTER = 4
N
CALL NERR
310
50
Y
50

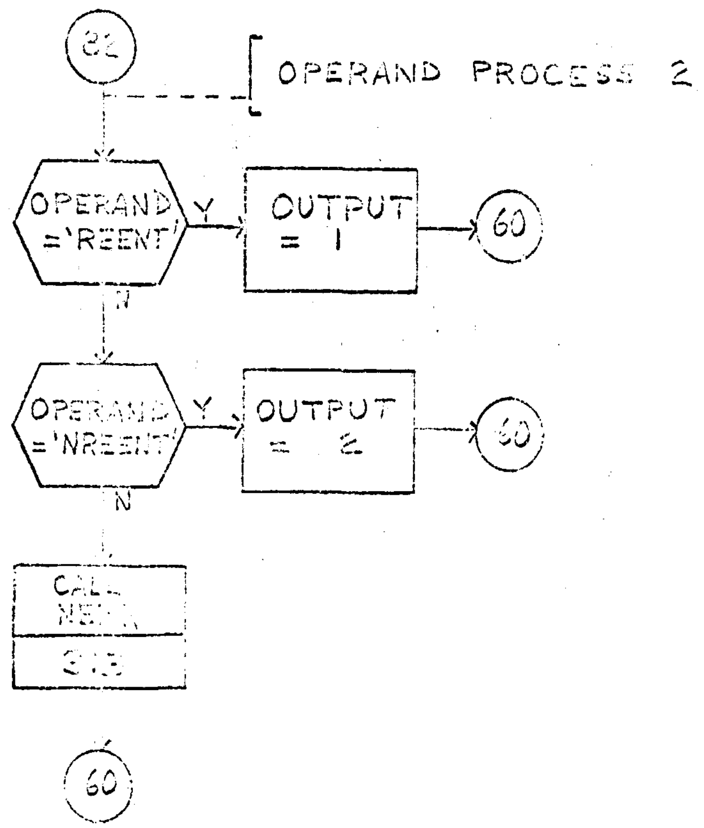

58
PRE-PROCESS 8 CHECKS
END CARD SEQUENCE
SEQUENCE CHECKING COUNTER = 4
N
CALL NERR
311
50
Y
SET COUNTER TO 0
50

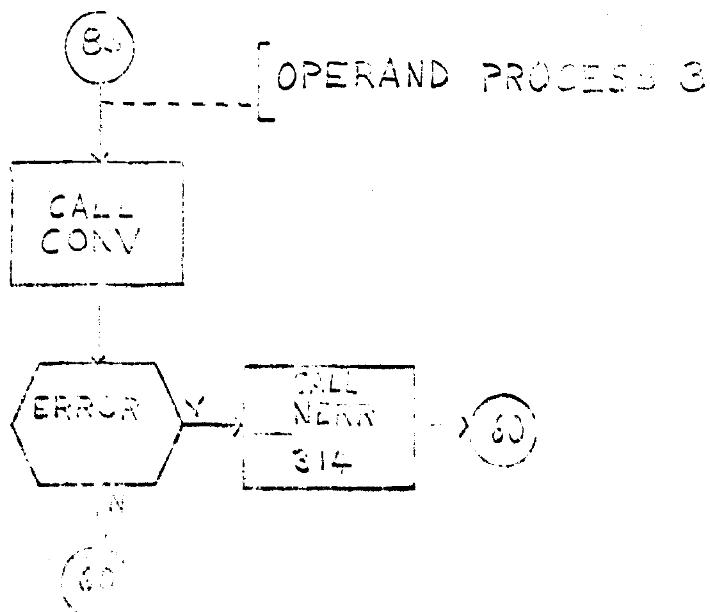


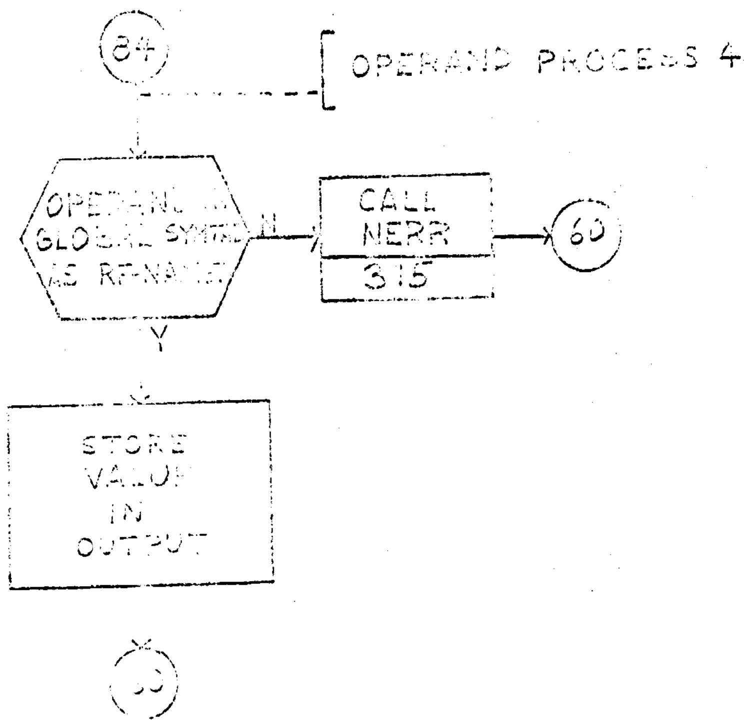
50
AFTER PRE-PROCESSING, PROCESS OPERANDS
IS THERE AN OPERAND TO BE PROCESSED
N
70
Y
IS STATEMENT OPERAND = 0
Y
61
N
IS TABLE OPERAND < 0
Y
62
N
SET J EQUAL TO THE TABLE OPERAND
63

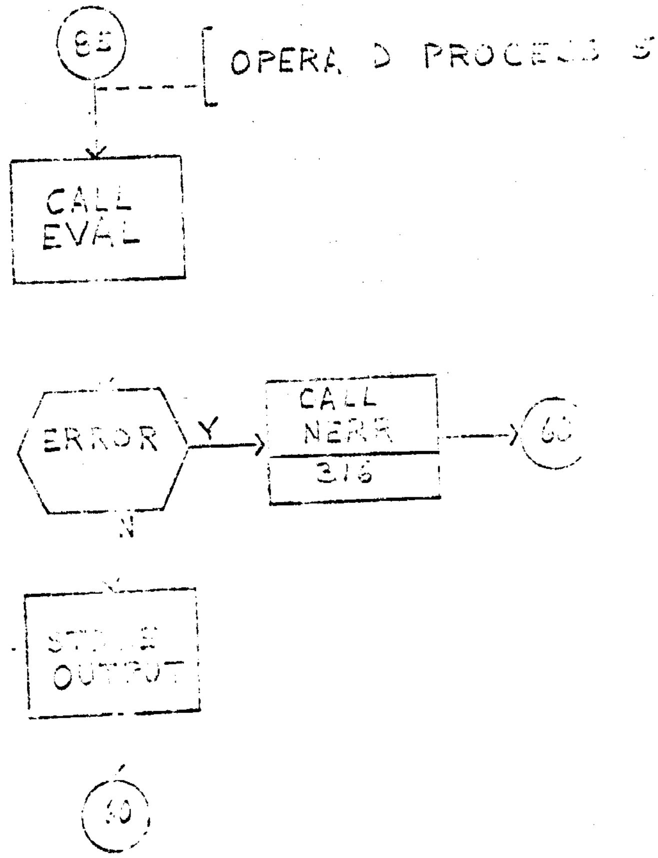

62
SET J EQUAL TO THE TABLE OPERAND WITH THE SIGN REVERSED
63
J EQUALS O
N
GO TO THE "J"TH OPERAND PROCESS
Y
60
HAVE ALL SIX OPERANDS BEEN PROCESSED
N
50
Y
70
POST-PROCESSING IS REQUIRED
Y
GO TO POST-PROCESS
N
150

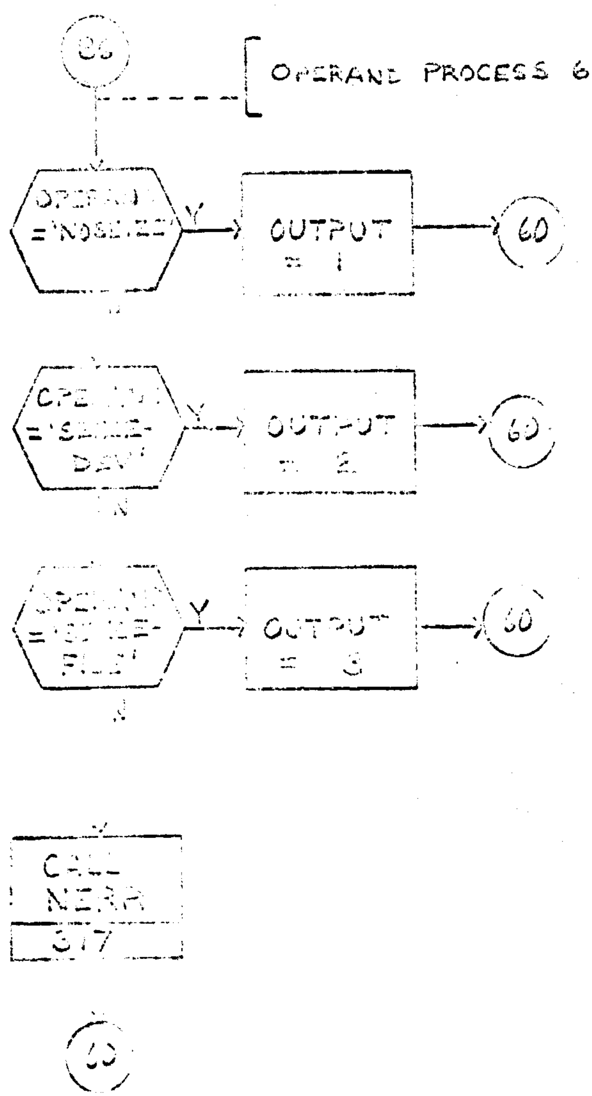

81
OPERAND PROCESS
OPERAND = 'OS'
Y
N
OUTPUT = 1
60
OPERAND = 'PW'
Y
N
OUTPUT = 2
60
OPERAND = 'COMM'
Y
N
OUTPUT = 3
60
OPERAND = 'SCI'
Y
N
OUTPUT = 4
60
CALL NEHR
60

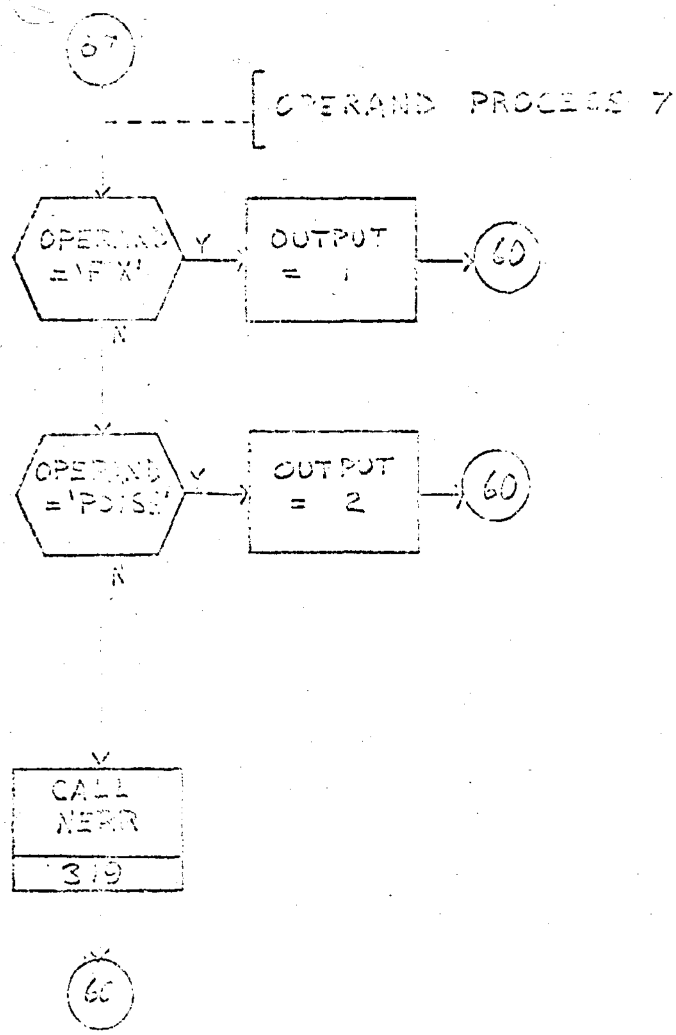

82
OPERAND PROCESS 2
OPERAND ='REENT'
Y
OUTPUT = 1
60
N
OPERAND ='NREENT'
Y
OUTPUT = 2
60
N
CALL NERR
60

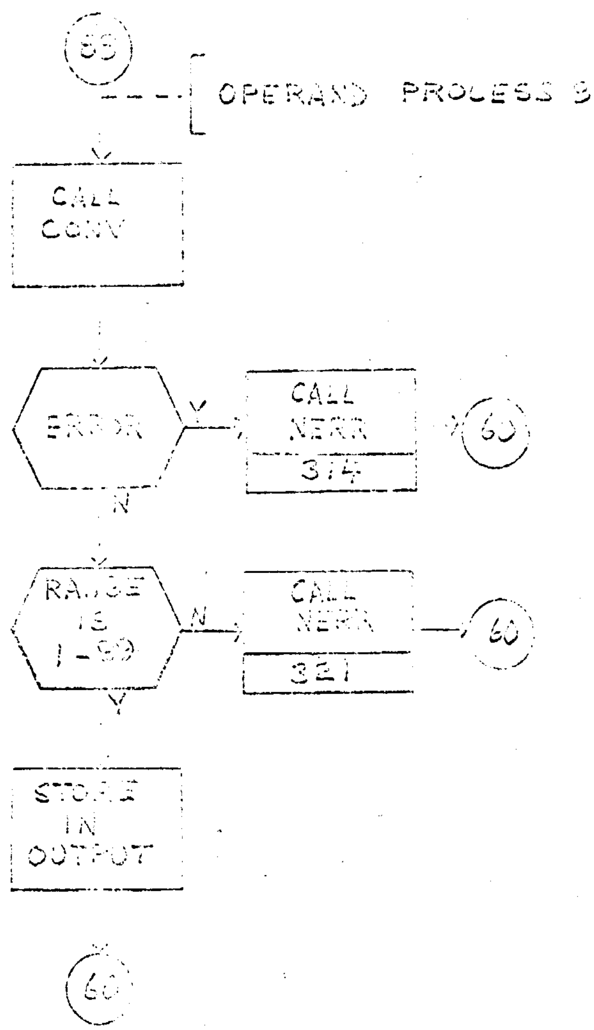

OPERAND PROCESS 3
CALL CONV
ERROR
Y
CALL NERR
314
60
N

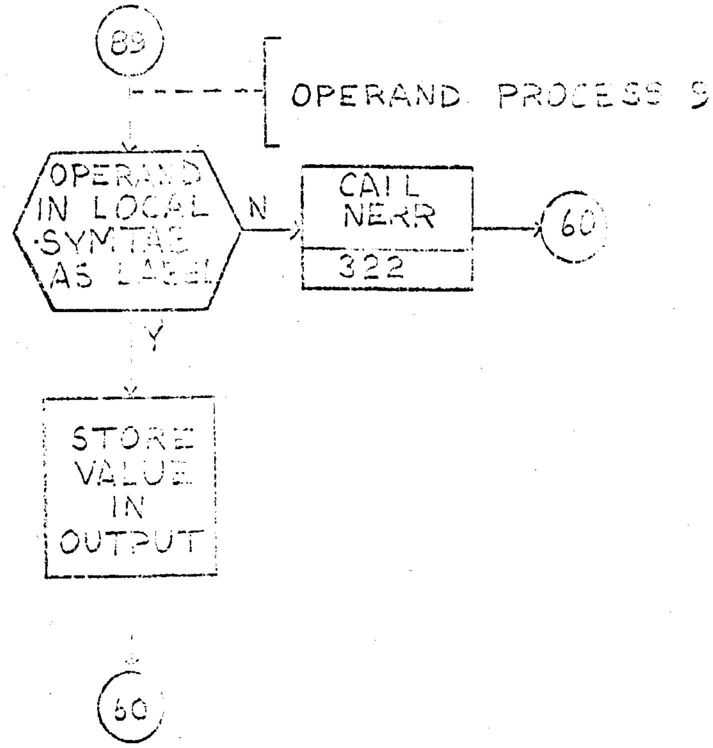

84
OPERAND PROCESS 4
OPERAND IN GLOBAL SYMTAB AS RFNAME?
N
CALL NERR
315
60
Y
STORE VALUE IN OUTPUT

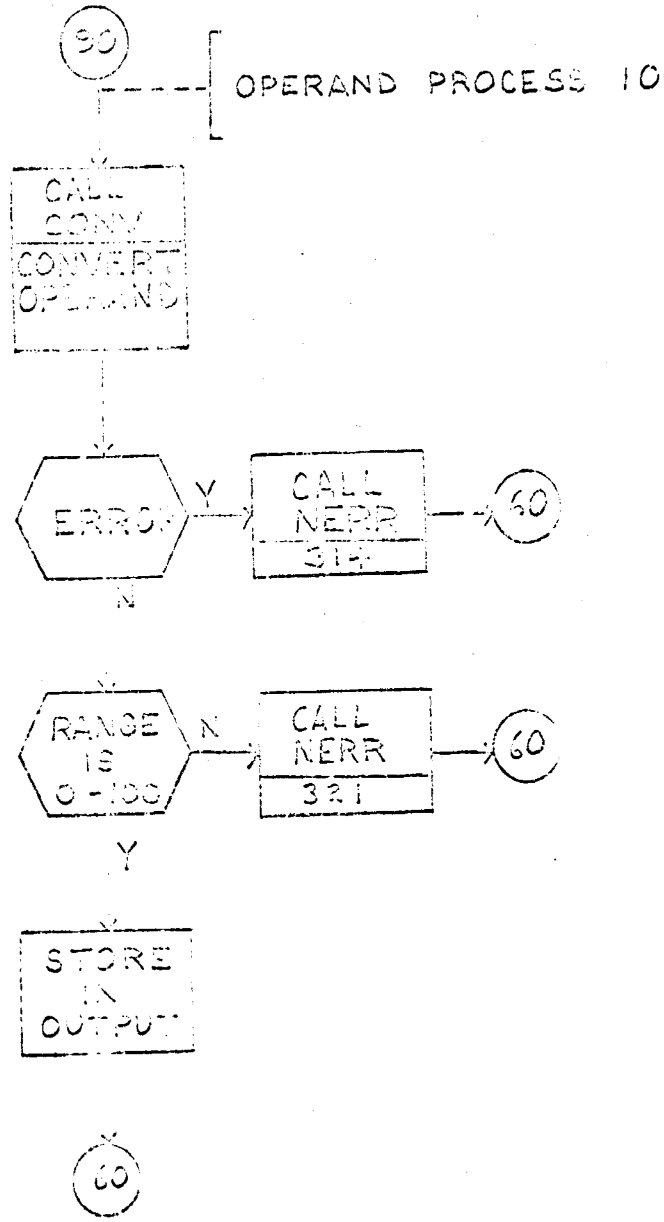

CALL
EVAL
ERROR
Y
N
CALL
NERR
316
OUTPUT

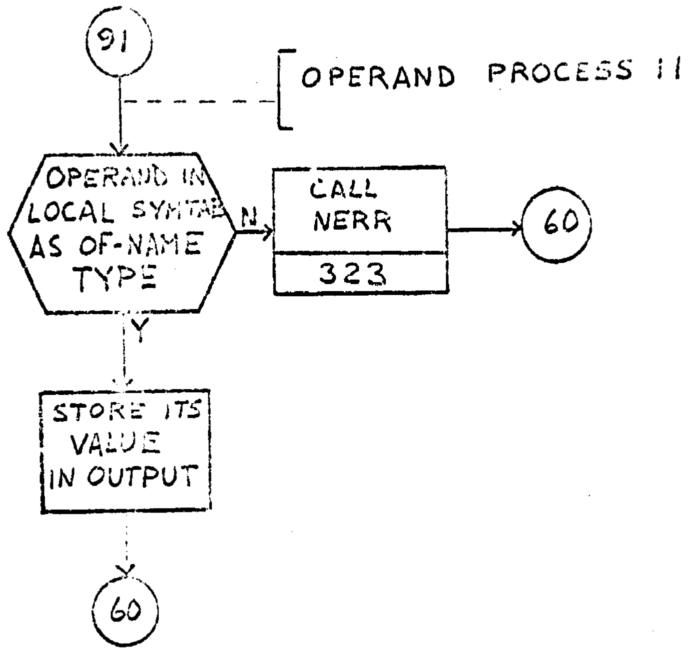

OPERAND PROCESS 6
='NOSEIZE'
OUTPUT
= 1
60
OUTPUT
= 2
60
OUTPUT
= 3
60
CALL
NERR
317

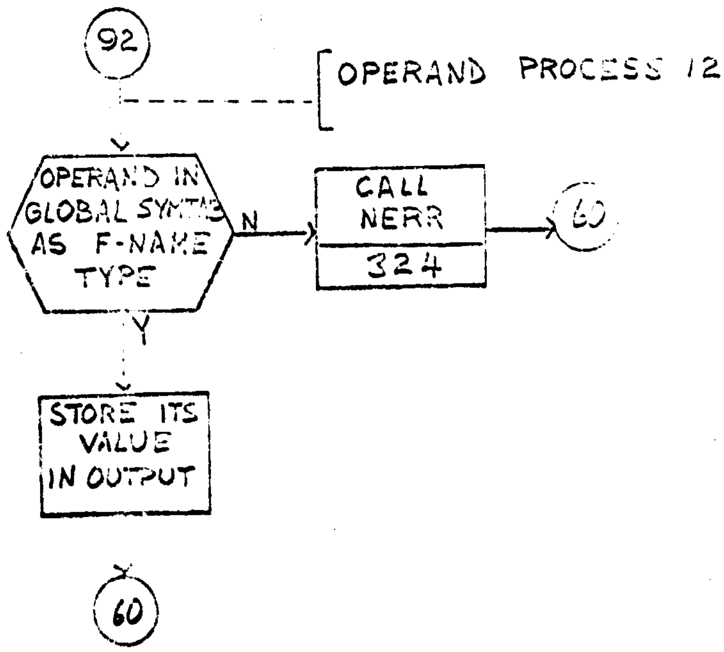

67
OPERAND PROCESS 7
OPERAND ='FIX'
Y
N
OUTPUT = 1
60
OPERAND ='POISE'
Y
N
OUTPUT = 2
60
CALL NERR
319
60

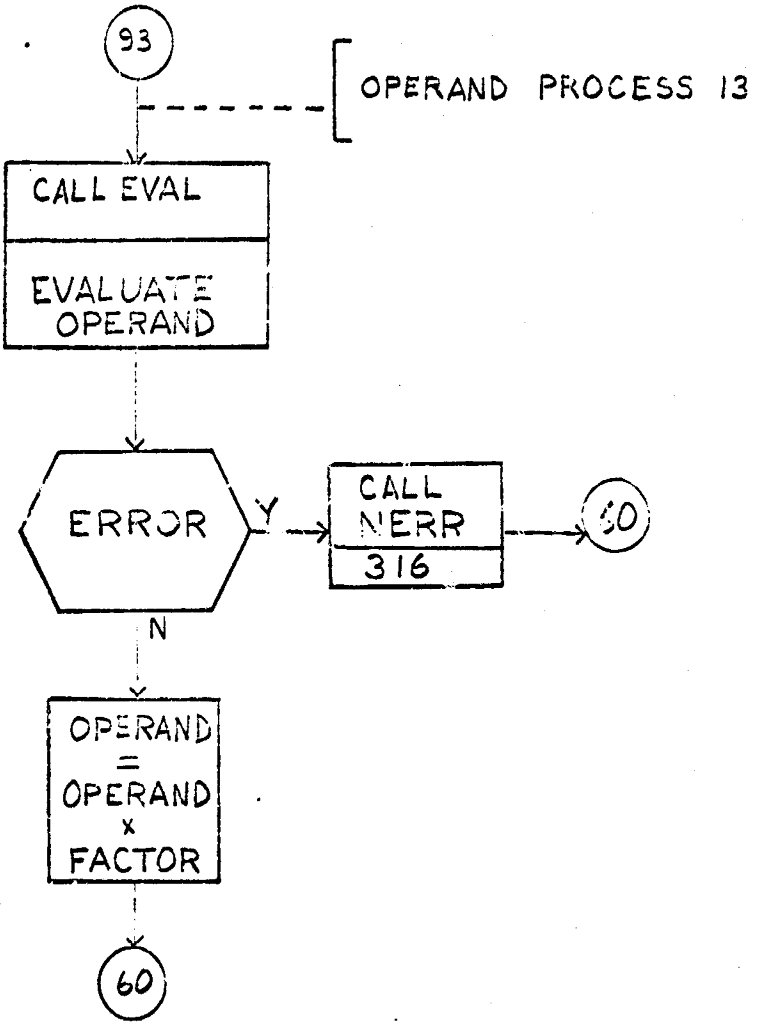

OPERAND PROCESS
CALL CONV
ERROR
Y
CALL NERR
374
60
N
RANGE IS 1-99
N
CALL NERR
321
60
Y
STORE IN OUTPUT
60

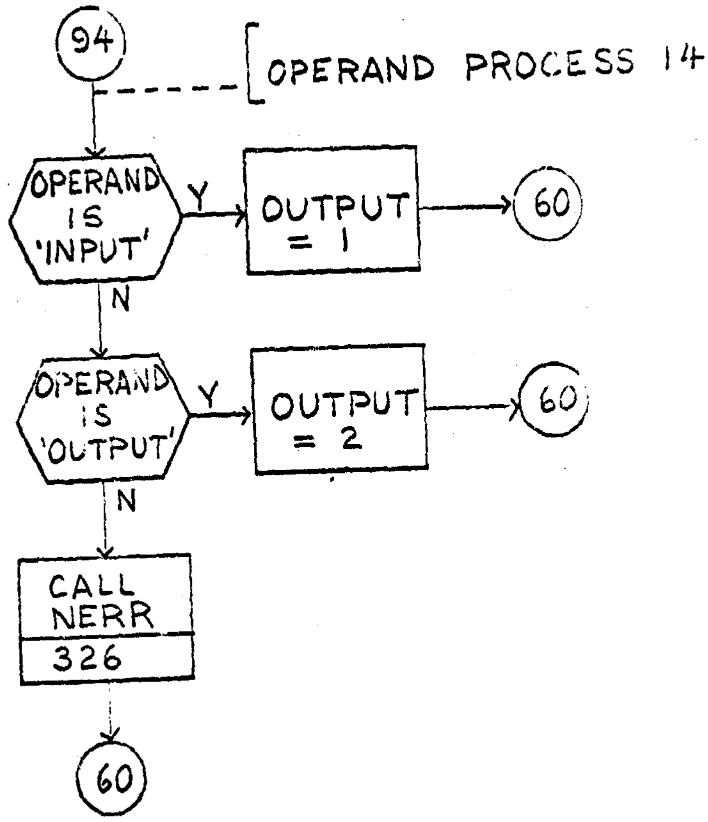

89
OPERAND PROCESS 9
OPERAND IN LOCAL SYMTAB AS LABEL
N
CALL NERR
322
60
Y
STORE VALUE IN OUTPUT
60

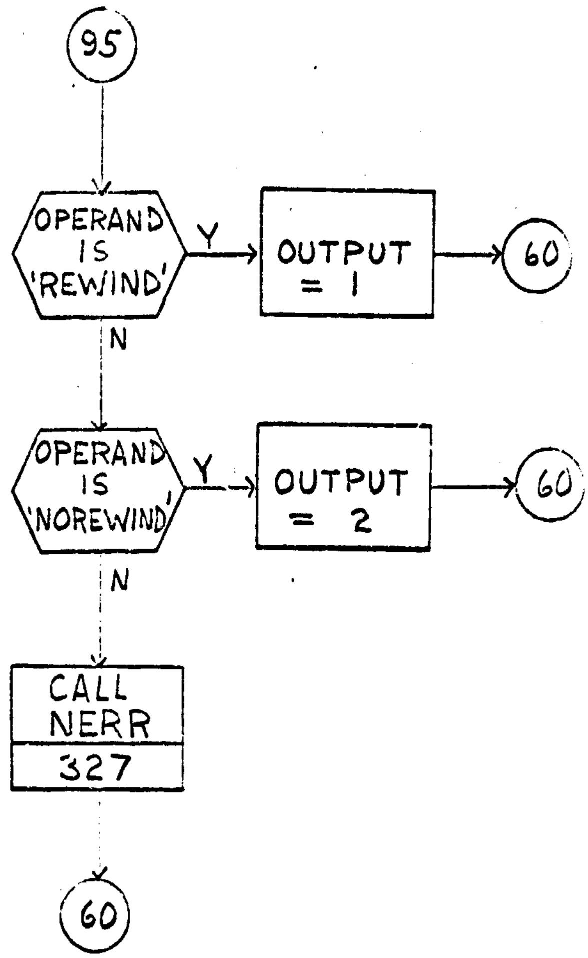

90
OPERAND PROCESS 10
CALL CONV
CONVERT OPERAND
ERROR
Y
N
CALL NERR
314
60
RANGE IS 0-100
N
Y
CALL NERR
321
60
STORE IN OUTPUT
60

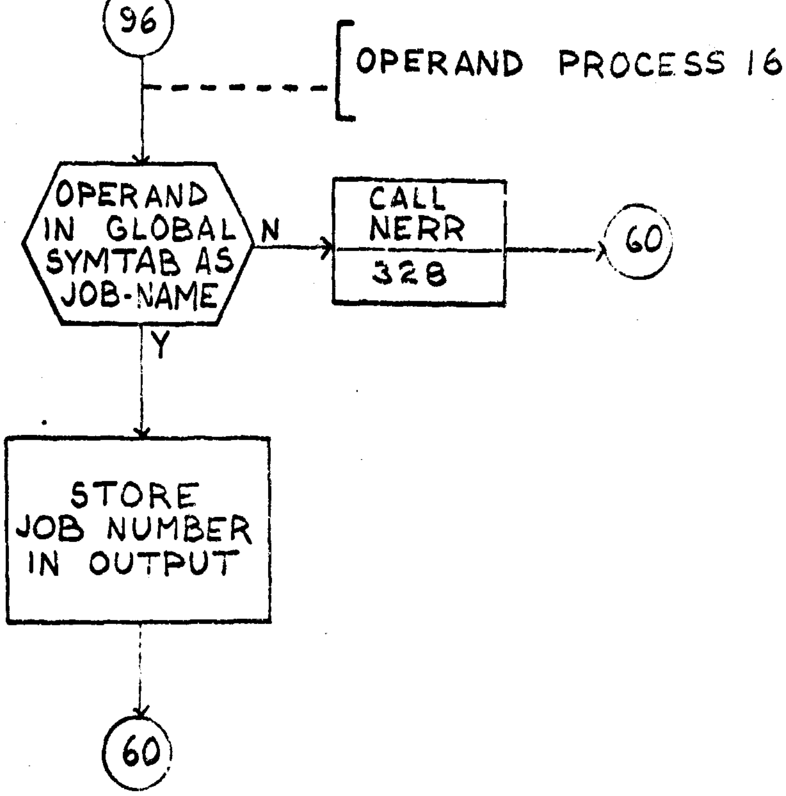

91
OPERAND PROCESS 11
OPERAND IN LOCAL SYMTAB AS OF-NAME TYPE
N
CALL NERR
323
60
Y
STORE ITS VALUE IN OUTPUT
60

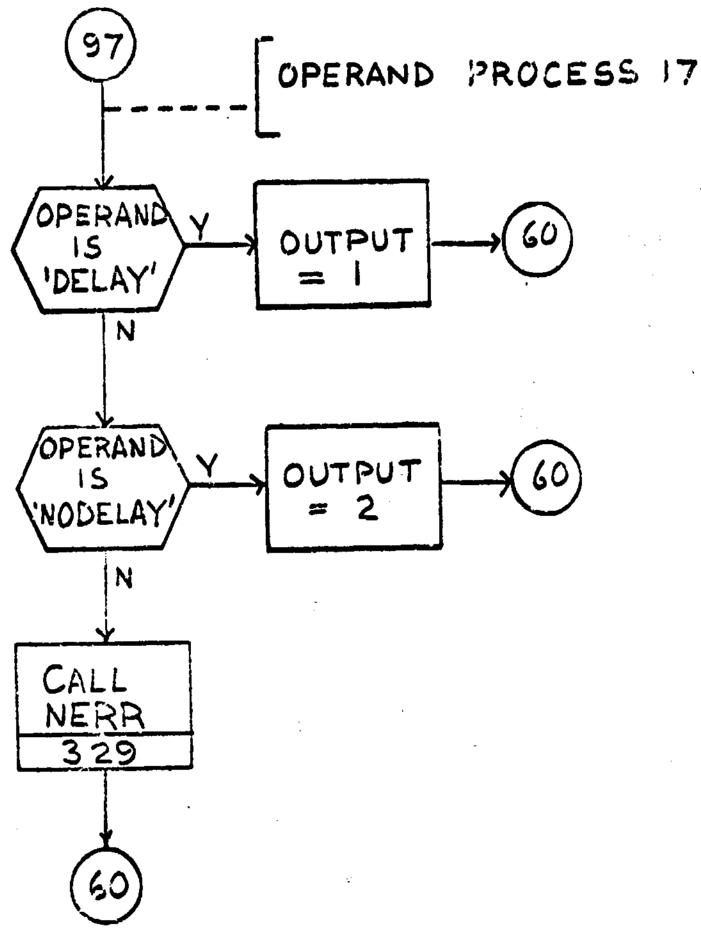

92
OPERAND PROCESS 12
OPERAND IN GLOBAL SYMTAB AS F-NAME TYPE
N
CALL NERR
324
60
Y
STORE ITS VALUE IN OUTPUT
60

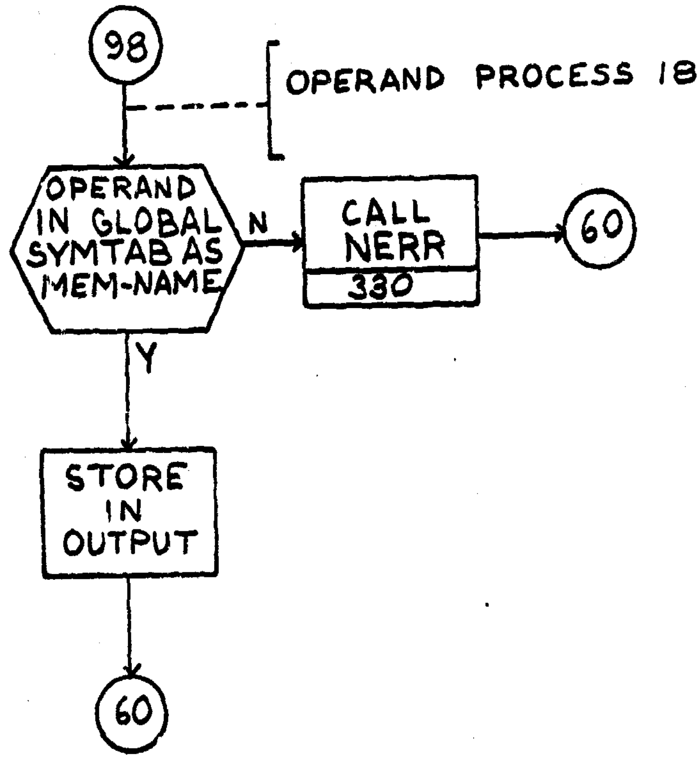

93
OPERAND PROCESS 13
CALL EVAL
EVALUATE OPERAND
ERROR
Y
CALL NERR
316
60
N
OPERAND = OPERAND x FACTOR
60

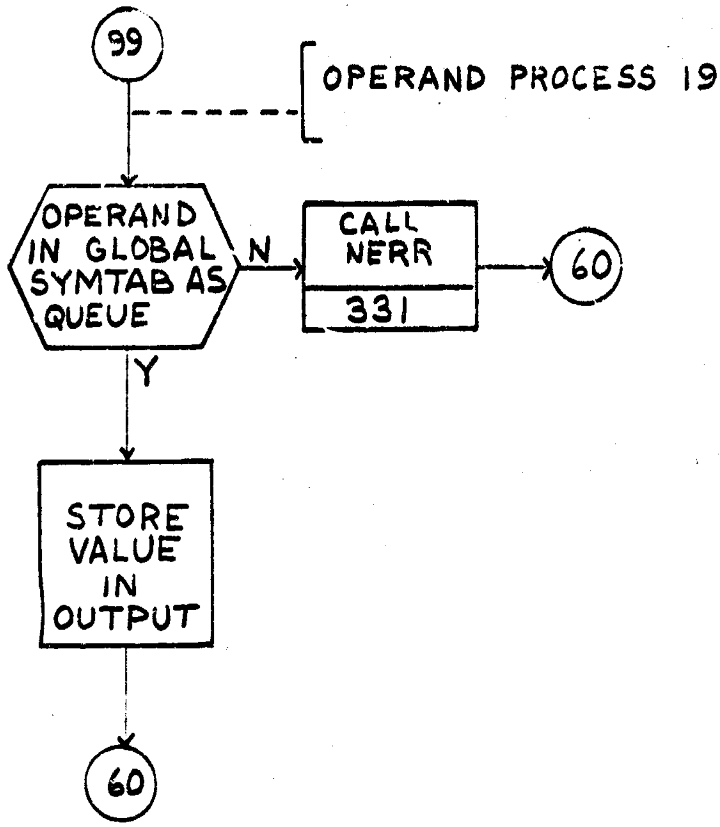

94
OPERAND PROCESS 14
OPERAND IS 'INPUT'
Y
N
OUTPUT = 1
60
OPERAND IS 'OUTPUT'
Y
N
OUTPUT = 2
60
CALL NERR
326
60

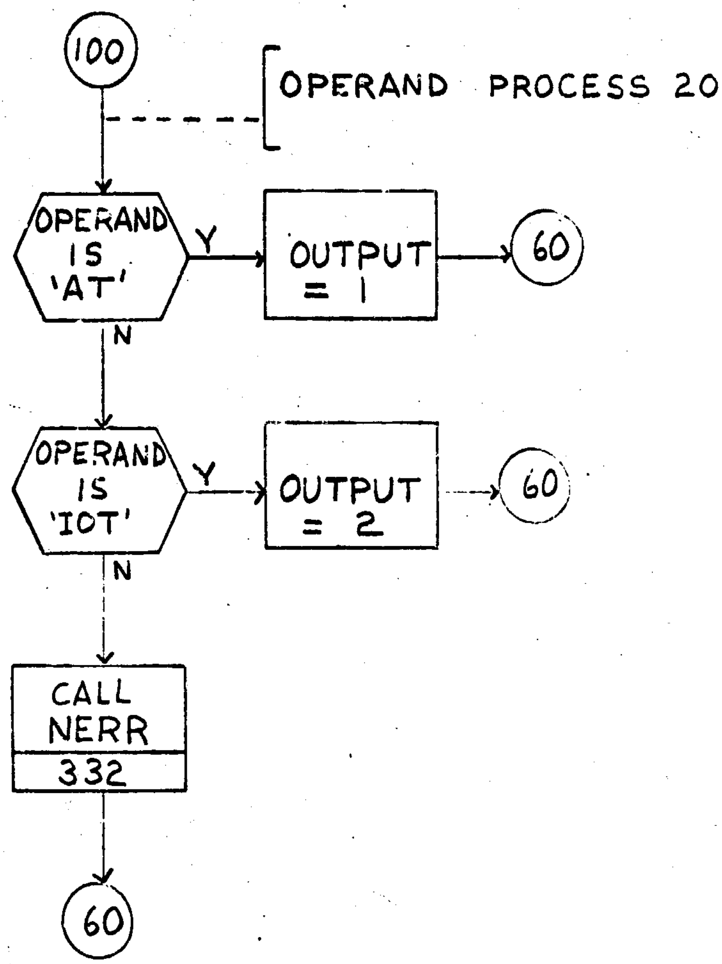

95
OPERAND IS 'REWIND'
Y
OUTPUT = 1
60
N
OPERAND IS 'NOREWIND'
Y
OUTPUT = 2
60
N
CALL NERR
327
60

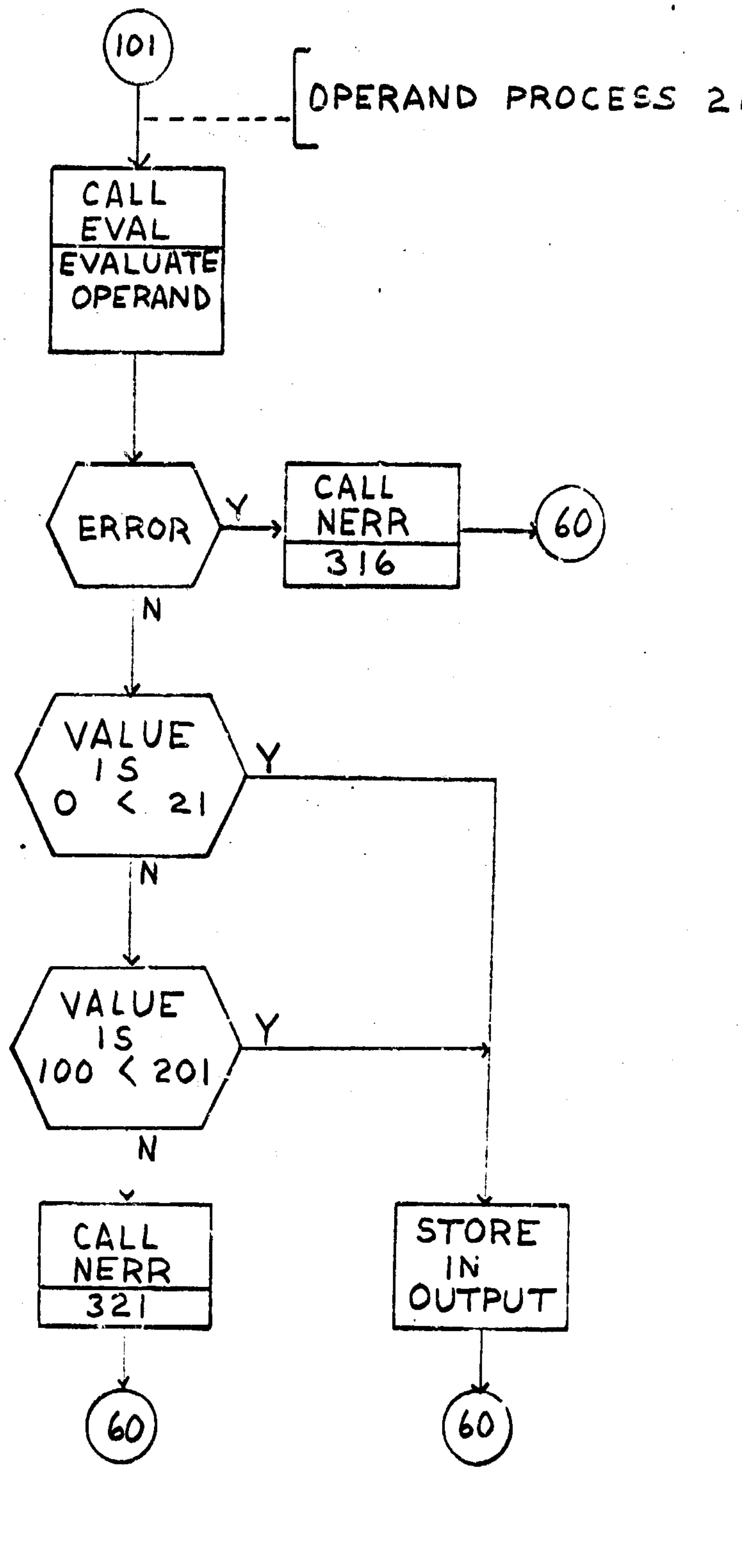

96
OPERAND PROCESS 16
OPERAND IN GLOBAL SYMTAB AS JOB-NAME
N
CALL NERR
328
60
Y
STORE JOB NUMBER IN OUTPUT
60

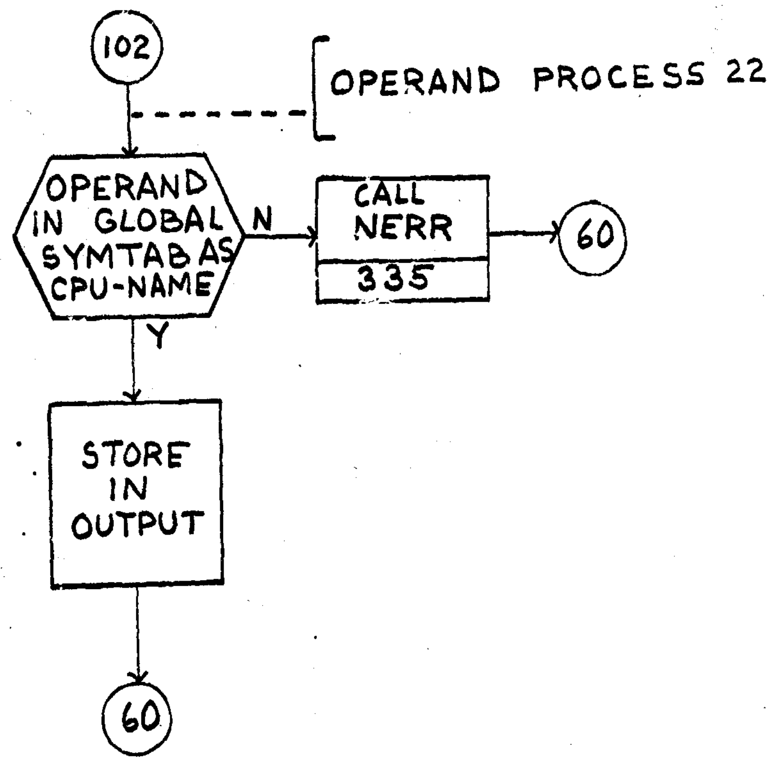

97
OPERAND PROCESS 17
OPERAND IS 'DELAY'
Y
OUTPUT = 1
60
N
OPERAND IS 'NODELAY'
Y
OUTPUT = 2
60
N
CALL NERR
329
60

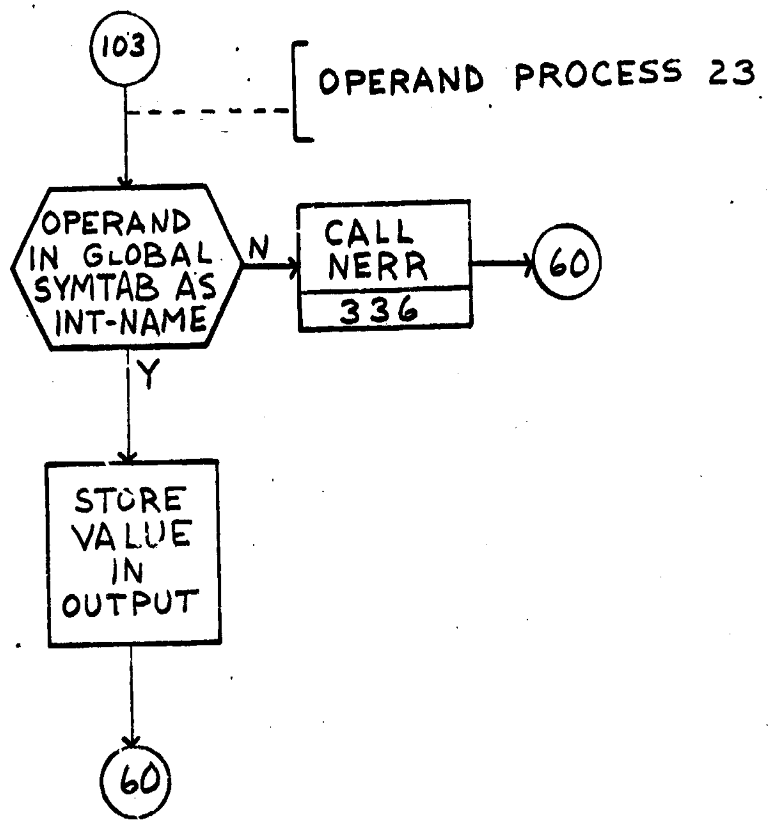

98
OPERAND PROCESS 18
OPERAND IN GLOBAL SYMTAB AS MEM-NAME
N
CALL NERR
330
60
Y
STORE IN OUTPUT
60

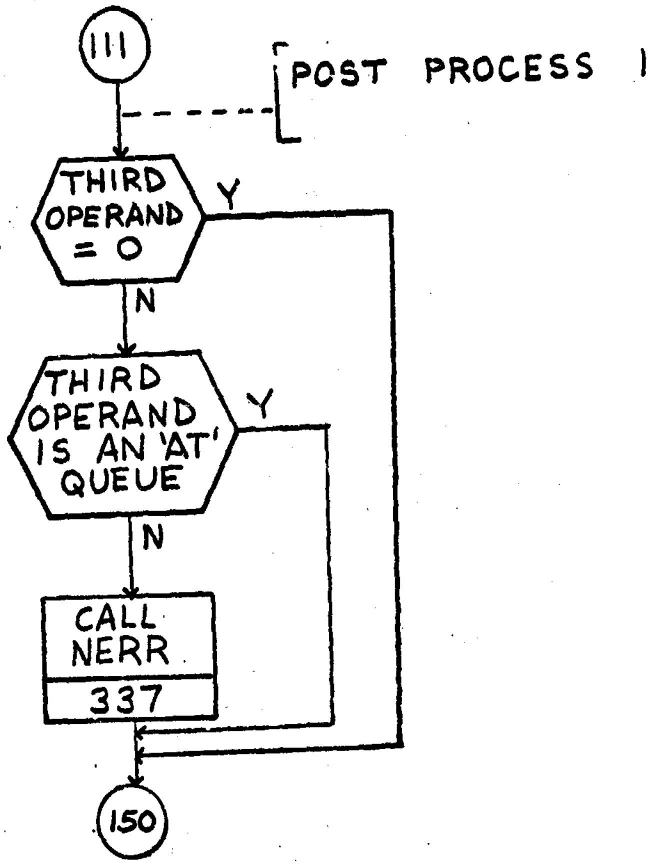

99
OPERAND PROCESS 19
OPERAND IN GLOBAL SYMTAB AS QUEUE
N
Y
CALL NERR
331
60
STORE VALUE IN OUTPUT
60

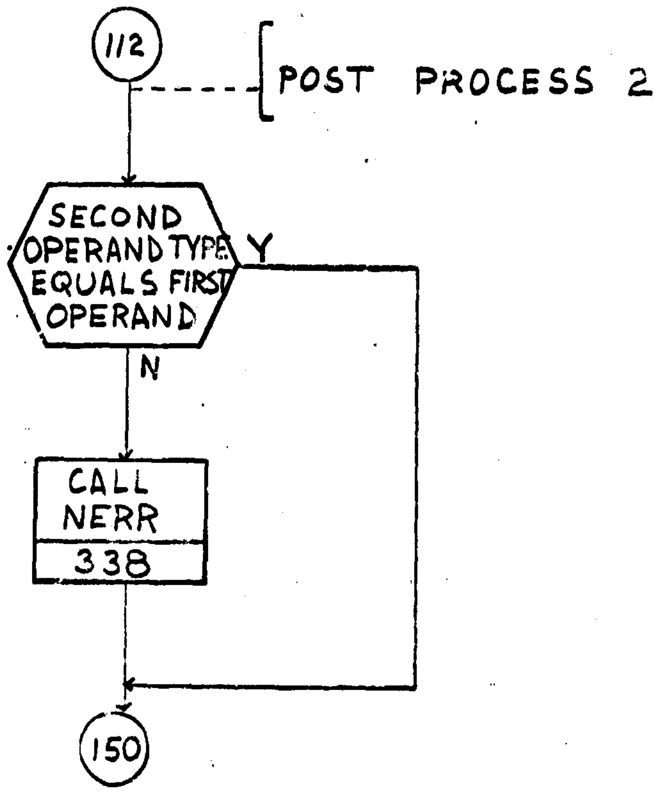

100
OPERAND PROCESS 20
OPERAND IS 'AT'
Y
N
OUTPUT = 1
60
OPERAND IS 'IOT'
Y
N
OUTPUT = 2
60
CALL NERR
332
60

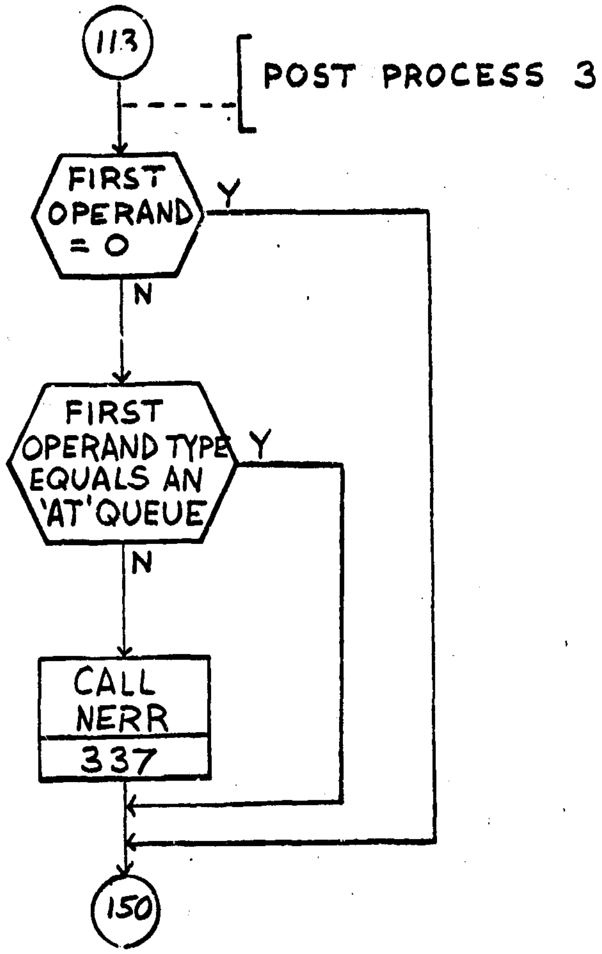

101
OPERAND PROCESS 21
CALL
EVAL
EVALUATE
OPERAND
ERROR
Y
CALL
NERR
316
60
N
VALUE
IS
0 < 21
Y
N
VALUE
IS
100 < 201
Y
N
CALL
NERR
321
STORE
IN
OUTPUT
60
60

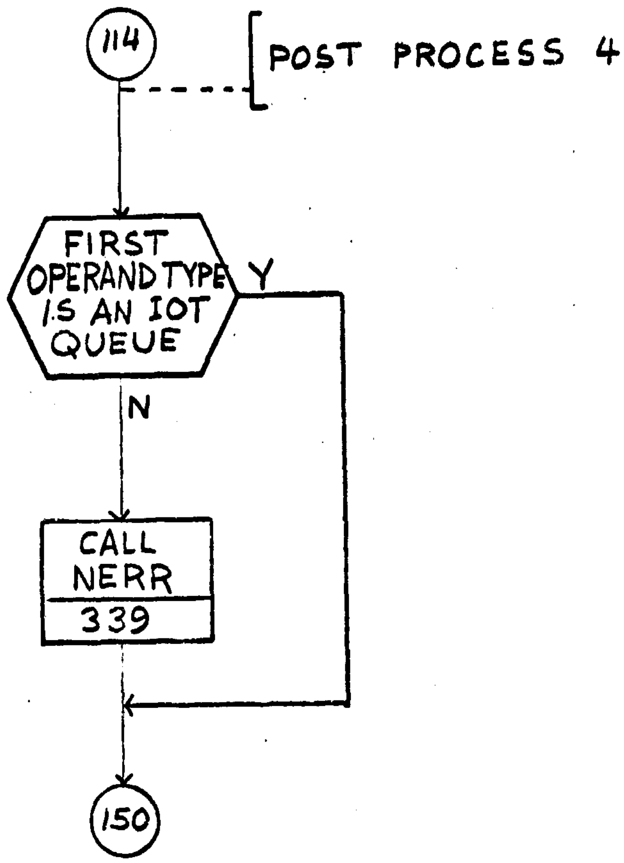

102
OPERAND PROCESS 22
OPERAND IN GLOBAL SYMTAB AS CPU-NAME
N
Y
CALL NERR
335
60
STORE IN OUTPUT
60

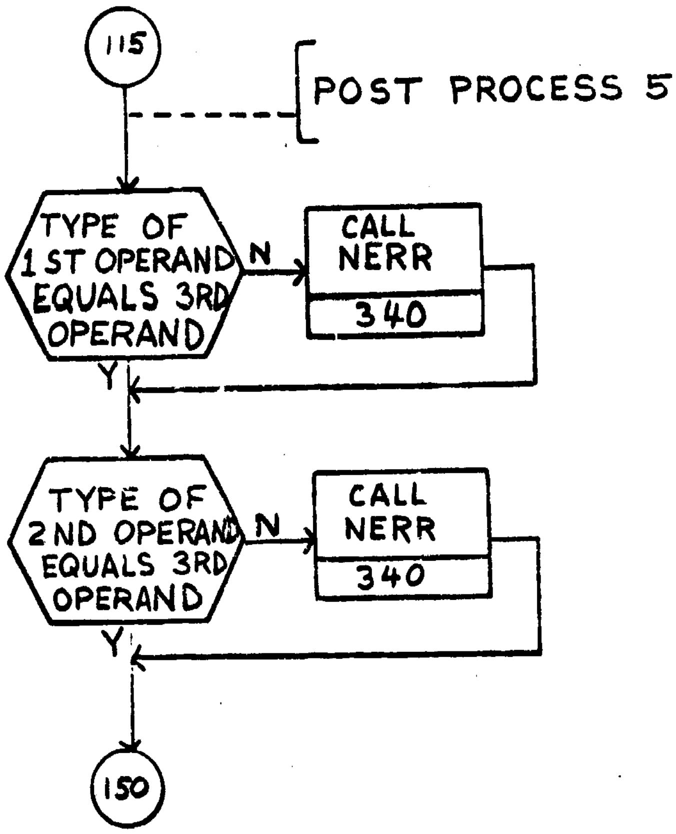

103
OPERAND PROCESS 23
OPERAND IN GLOBAL SYMTAB AS INT-NAME
N
CALL NERR
336
60
Y
STORE VALUE IN OUTPUT
60

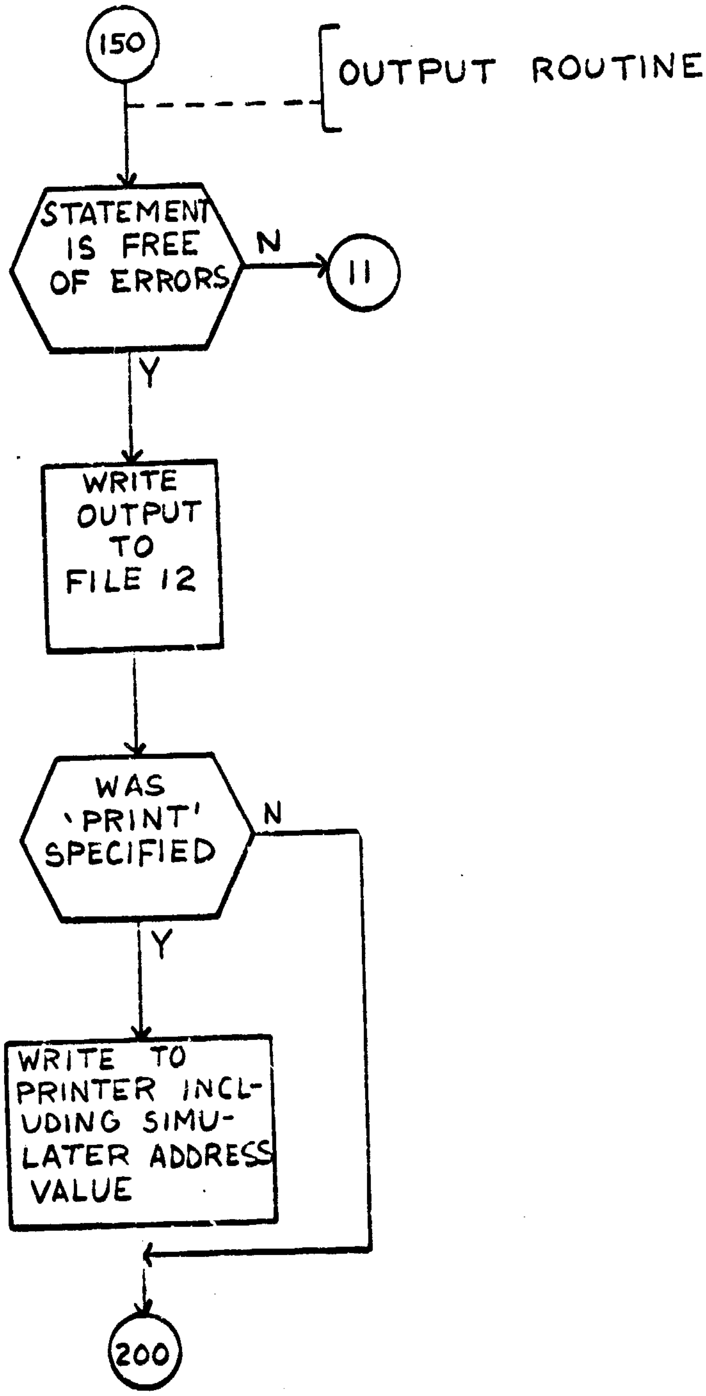

111
POST PROCESS 1
THIRD OPERAND = 0
Y
N
THIRD OPERAND IS AN 'AT' QUEUE
Y
N
CALL NERR
337
150

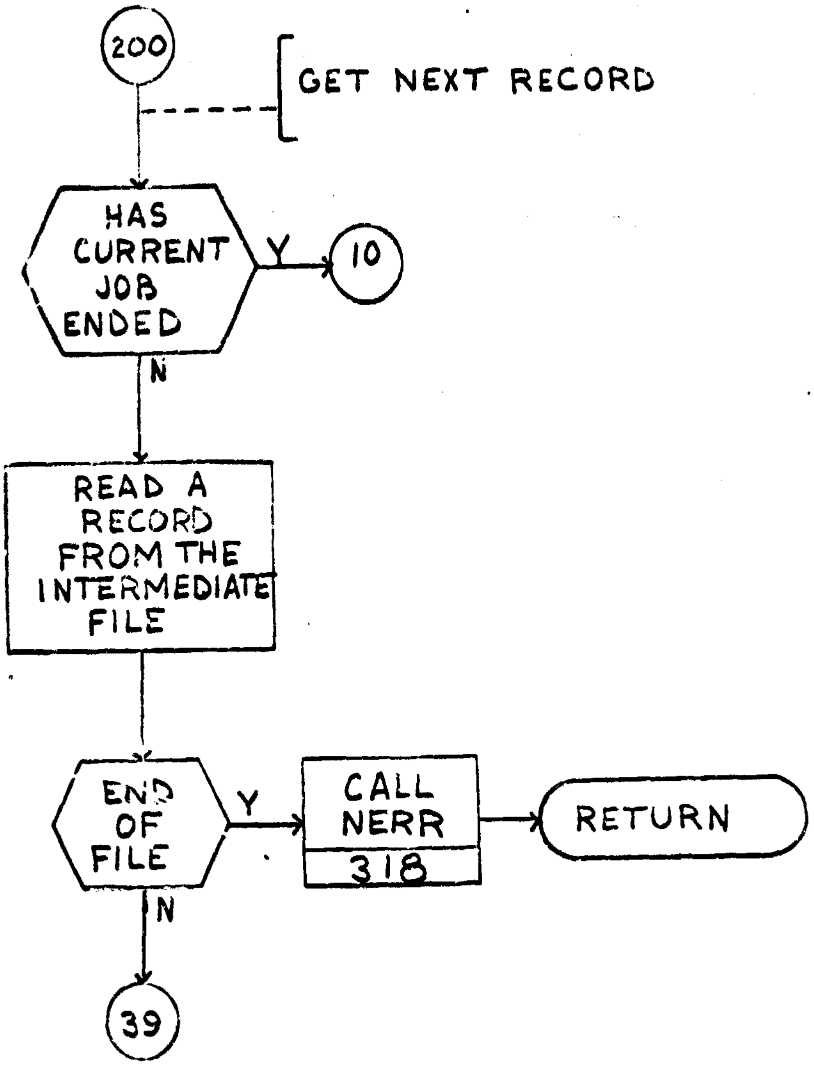

112
POST PROCESS 2
SECOND OPERAND TYPE EQUALS FIRST OPERAND
Y
N
CALL NERR
338
150


113
POST PROCESS 3
FIRST OPERAND = 0
Y
N
FIRST OPERAND TYPE EQUALS AN 'AT' QUEUE
Y
N
CALL NERR
337
150


114
POST PROCESS 4
FIRST OPERAND TYPE IS AN IOT QUEUE
Y
N
CALL NERR
339
150

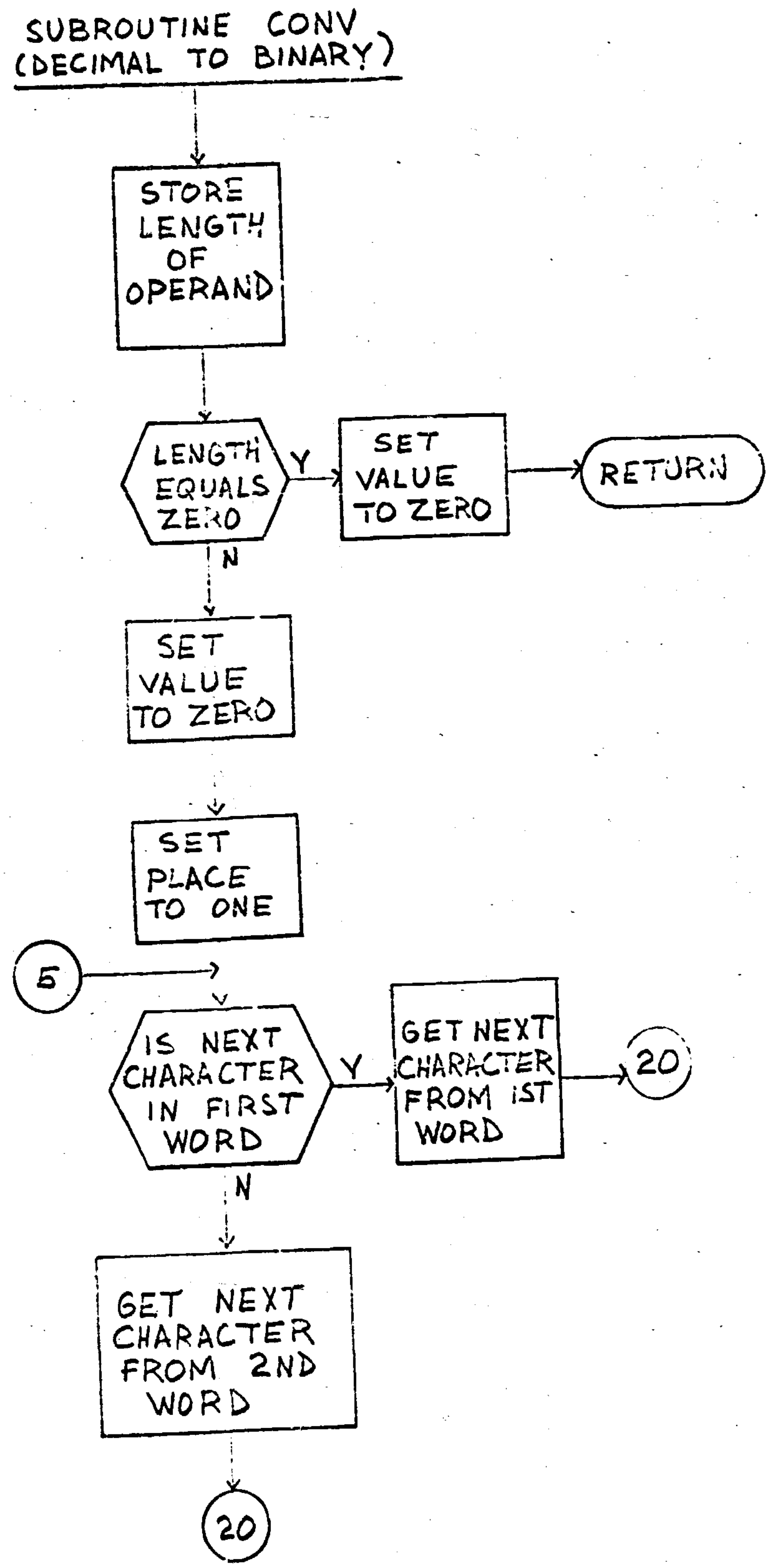

115
POST PROCESS 5
TYPE OF 1 ST OPERAND EQUALS 3RD OPERAND
N
CALL NERR
340
Y
TYPE OF 2 ND OPERAND EQUALS 3RD OPERAND
N
CALL NERR
340
Y
150

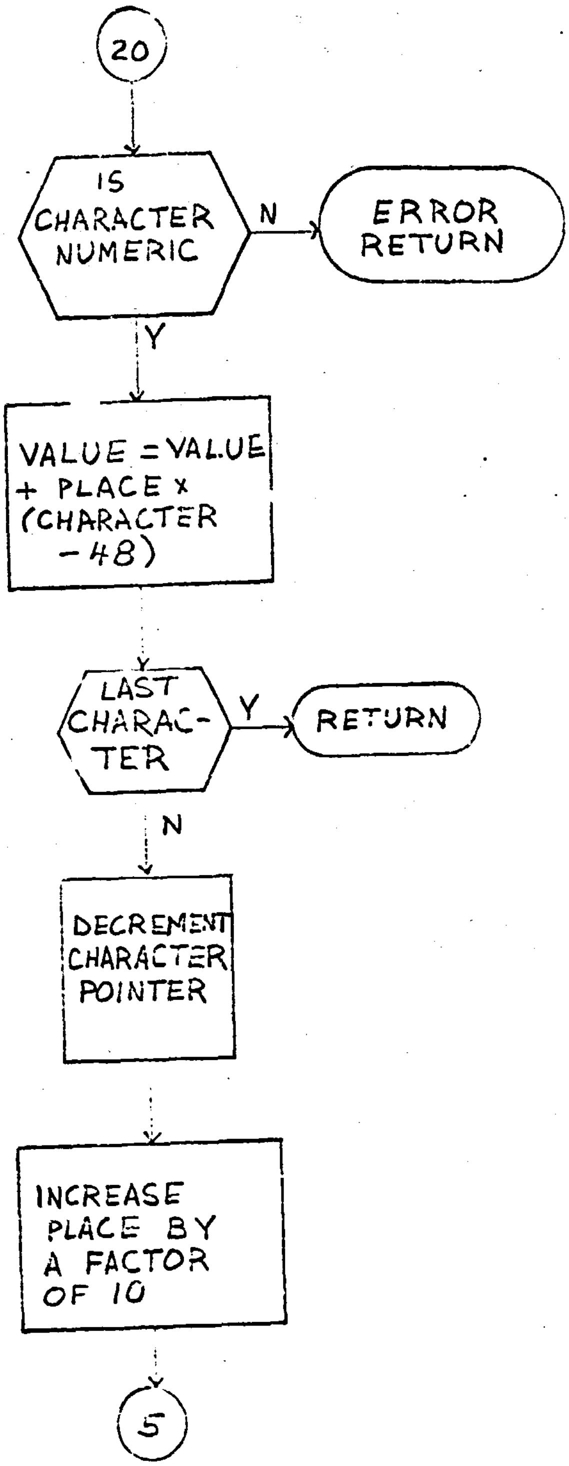

150
OUTPUT ROUTINE
STATEMENT IS FREE OF ERRORS
N
11
Y
WRITE OUTPUT TO FILE 12
WAS 'PRINT' SPECIFIED
N
Y
WRITE TO PRINTER INCL-UDING SIMU-LATER ADDRESS VALUE
200


200
GET NEXT RECORD
HAS CURRENT JOB ENDED
Y
10
N
READ A RECORD FROM THE INTERMEDIATE FILE
END OF FILE
Y
CALL NERR
318
RETURN
N
39

SUBROUTINE CONV

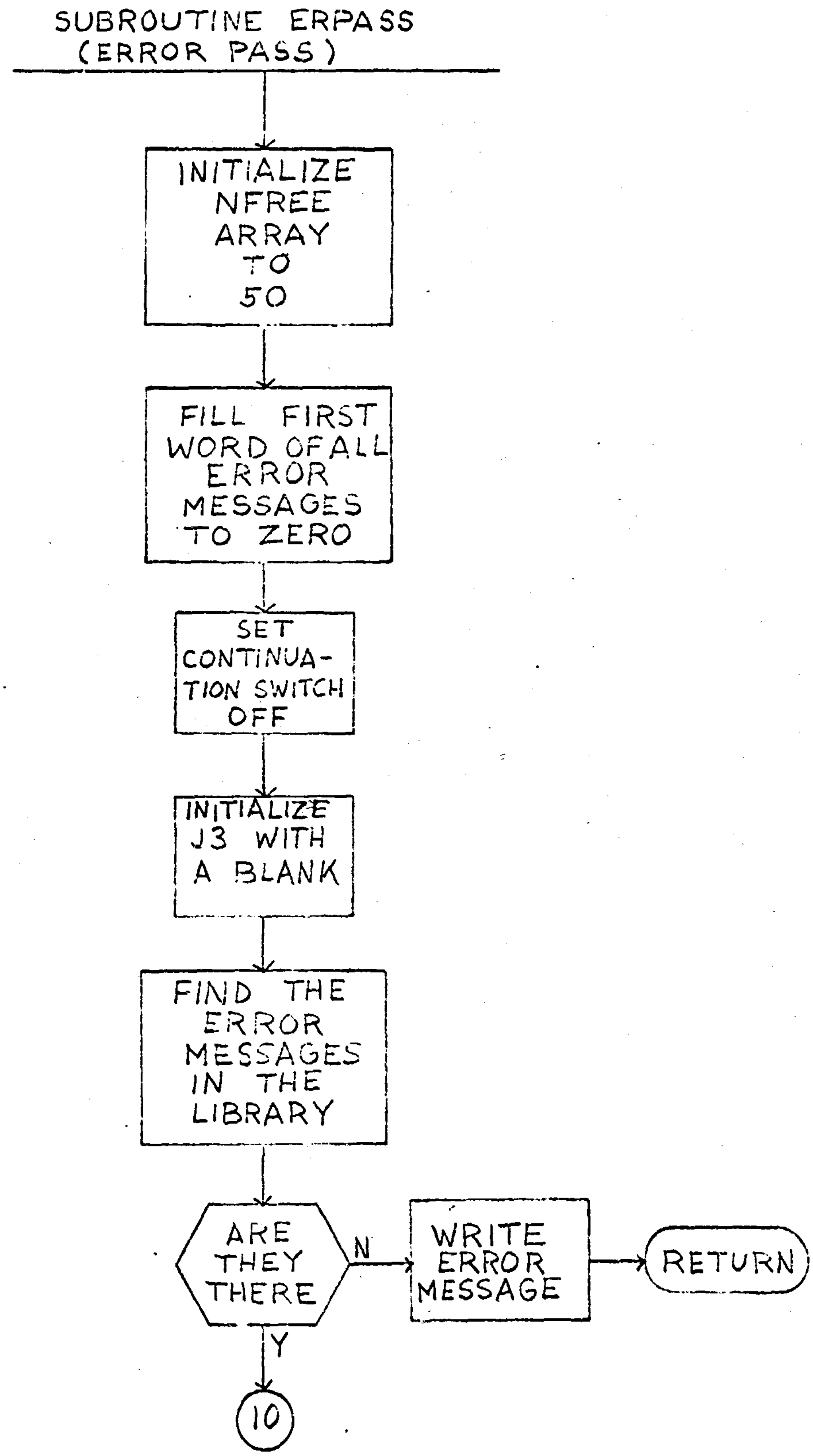
This is the routine which converts decimal representations of integers which are left justified in the fix field output format field into internal binary integer form. The routine is essentially a right to left scan of the decimal representation, in which each digit representation is converted to a binary value and multiplied by its positional value, and then added into a value. After all digits have been scanned, the net value, representing the binary value of the decimal representation, is returned.

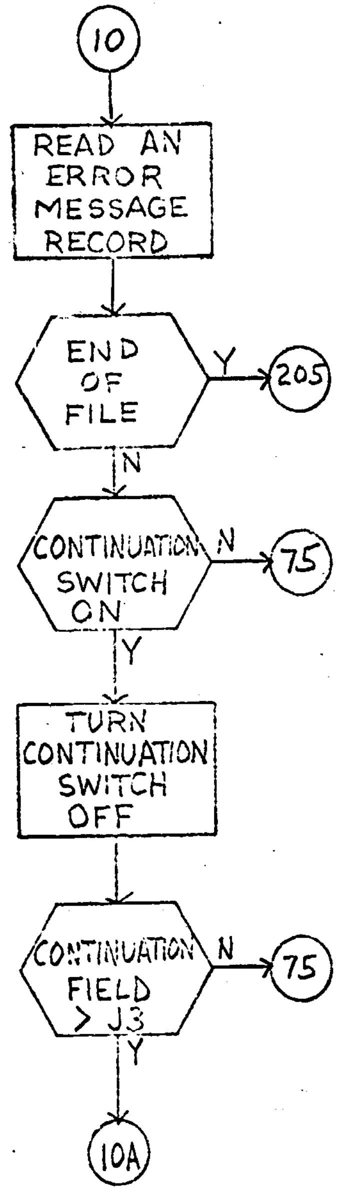

SUBROUTINE CONV
(DECIMAL TO BINARY)
STORE LENGTH OF OPERAND
LENGTH EQUALS ZERO
Y
N
SET VALUE TO ZERO
RETURN
SET VALUE TO ZERO
SET PLACE TO ONE
5
IS NEXT CHARACTER IN FIRST WORD
Y
N
GET NEXT CHARACTER FROM 1ST WORD
20
GET NEXT CHARACTER FROM 2ND WORD
20

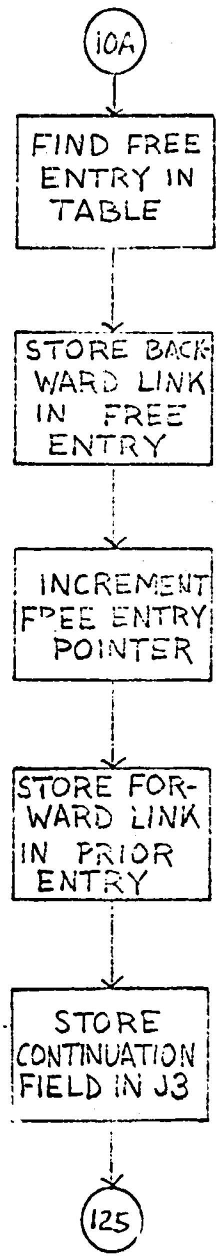

20
IS CHARACTER NUMERIC
N
ERROR RETURN
Y
VALUE = VALUE + PLACE x (CHARACTER - 48)
LAST CHARAC-TER
Y
RETURN
N
DECREMENT CHARACTER POINTER
INCREASE PLACE BY A FACTOR OF 10
5

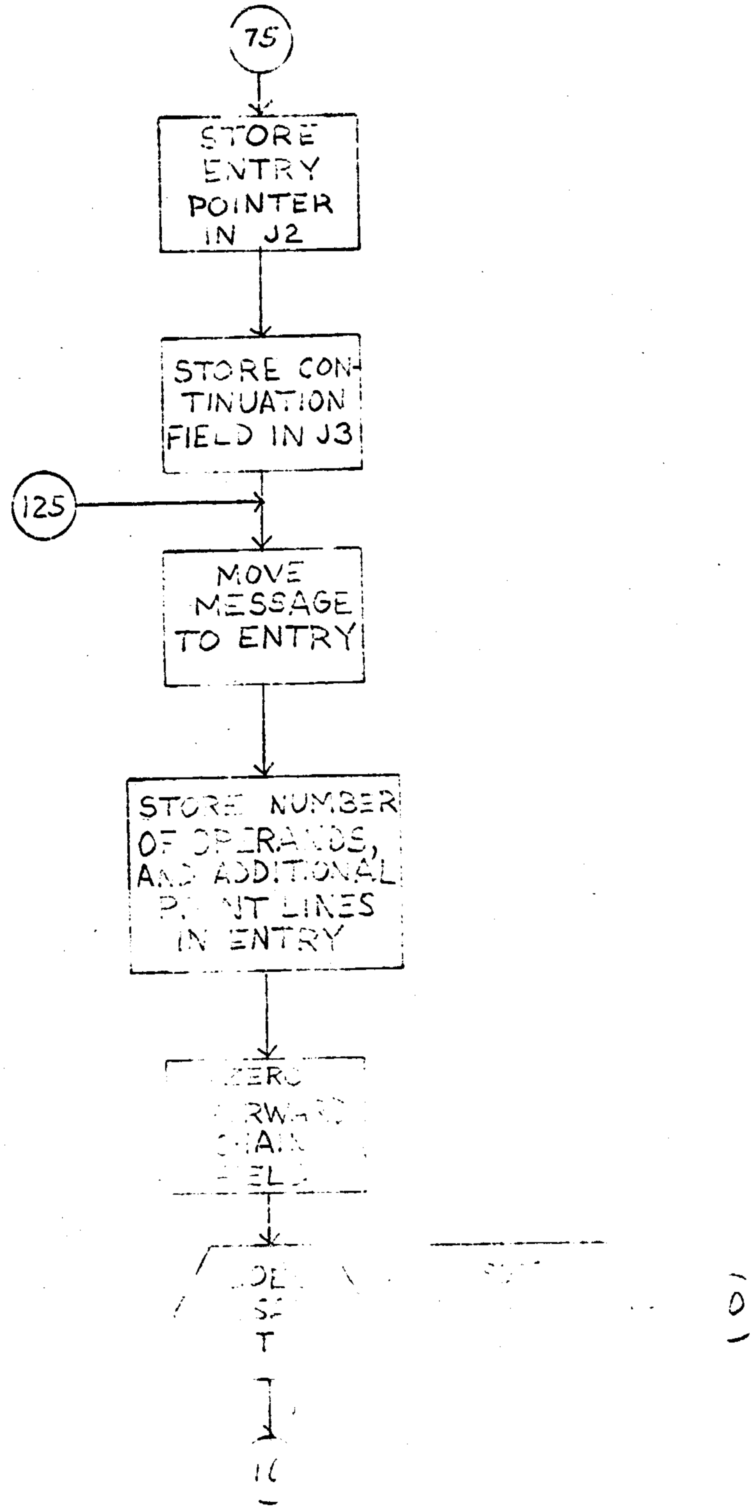
SUBROUTINE ERPASS

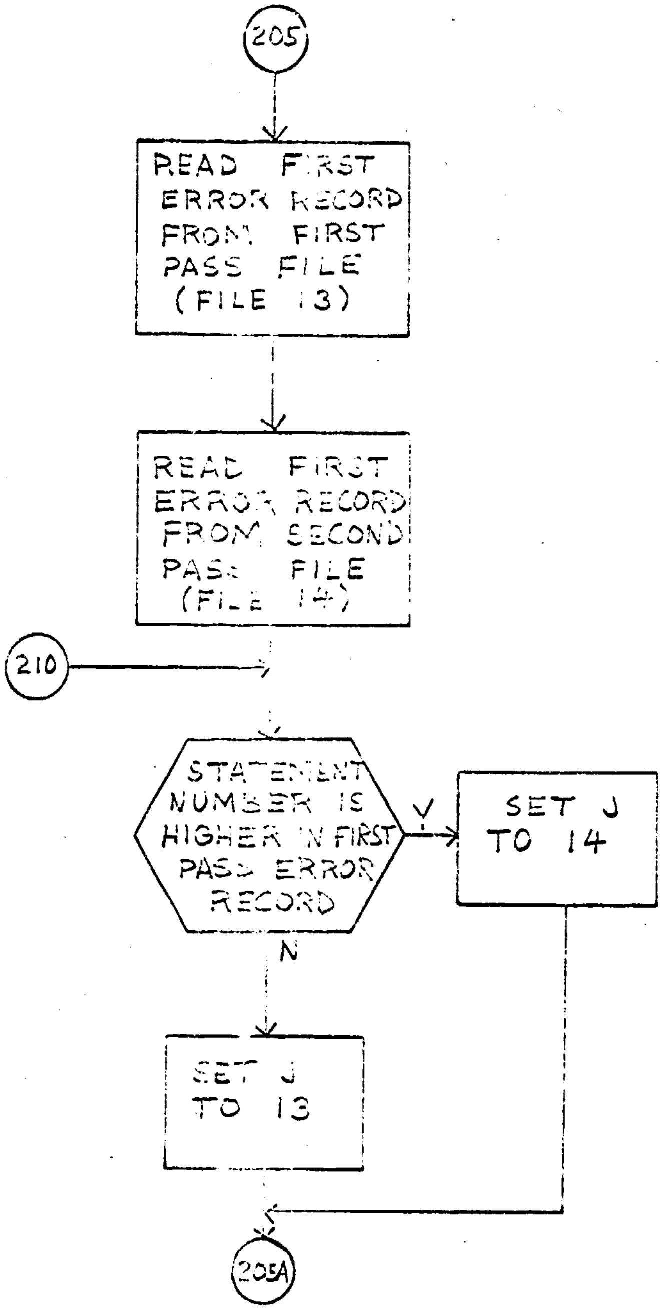
This routine merges the error files produced by the first and second passes and produces a listing of the errors in statement number sequence. For the sake of convenience, a prose diagnostic appears in addition to the statement number and error number.

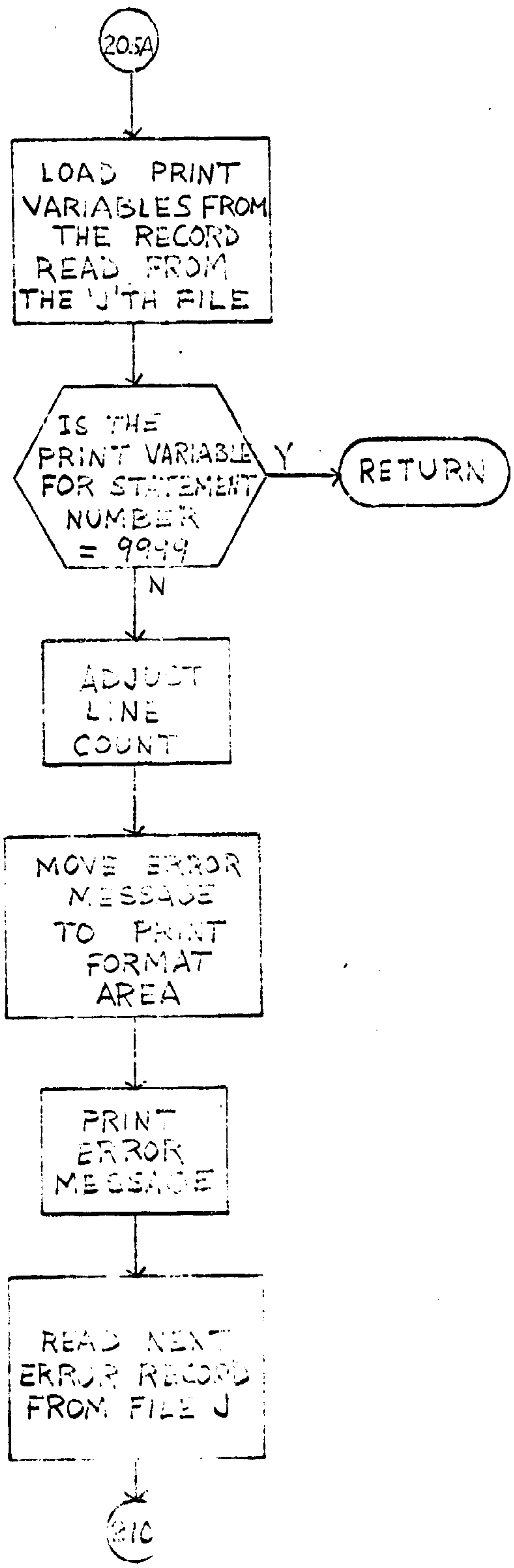

SUBROUTINE ERPASS
(ERROR PASS)
INITIALIZE NFREE ARRAY TO 50
FILL FIRST WORD OF ALL ERROR MESSAGES TO ZERO
SET CONTINUATION SWITCH OFF
INITIALIZE J3 WITH A BLANK
FIND THE ERROR MESSAGES IN THE LIBRARY
ARE THEY THERE
N
WRITE ERROR MESSAGE
RETURN
Y
10


10
READ AN ERROR MESSAGE RECORD
END OF FILE
Y
205
N
CONTINUATION SWITCH ON
N
75
Y
TURN CONTINUATION SWITCH OFF
CONTINUATION FIELD > J3
N
75
Y
10A


10A
FIND FREE ENTRY IN TABLE
STORE BACKWARD LINK IN FREE ENTRY
INCREMENT FREE ENTRY POINTER
STORE FORWARD LINK IN PRIOR ENTRY
STORE CONTINUATION FIELD IN J3
125


75
STORE ENTRY POINTER IN J2
STORE CONTINUATION FIELD IN J3
125
MOVE MESSAGE TO ENTRY
STORE NUMBER OF OPERANDS, AND ADDITIONAL
IN ENTRY
T

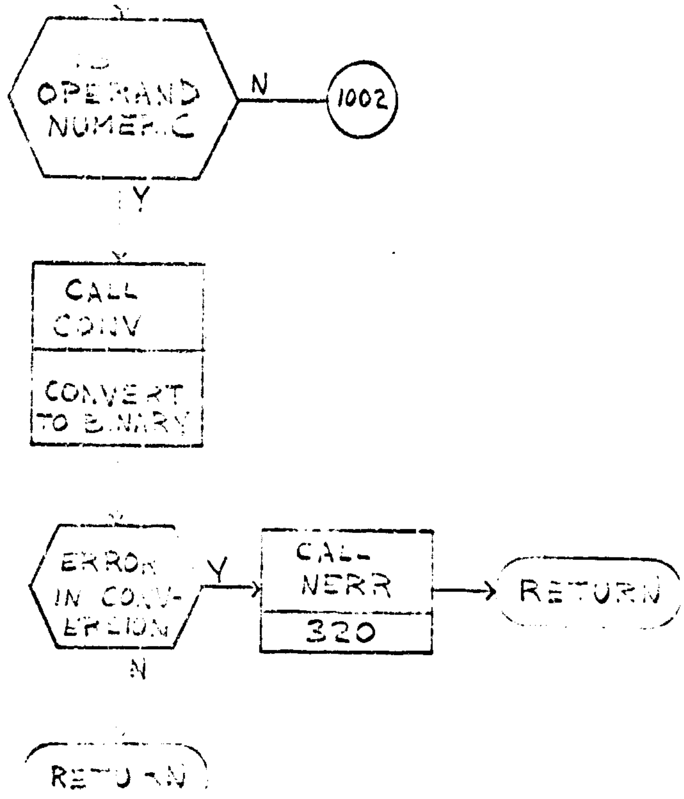

205
READ FIRST ERROR RECORD FROM FIRST PASS FILE (FILE 13)
READ FIRST ERROR RECORD FROM SECOND PASS FILE (FILE 14)
210
STATEMENT NUMBER IS HIGHER IN FIRST PASS ERROR RECORD
Y
SET J TO 14
N
SET J TO 13
205A

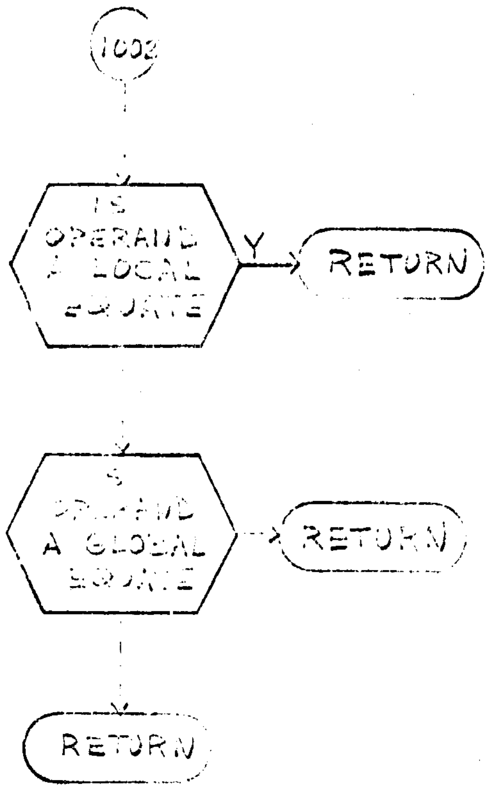

205A
LOAD PRINT VARIABLES FROM THE RECORD READ FROM THE 'J'TH FILE
IS THE PRINT VARIABLE FOR STATEMENT NUMBER = 9999
Y
RETURN
N
ADJUST LINE COUNT
MOVE ERROR MESSAGE TO PRINT FORMAT AREA
PRINT ERROR MESSAGE
READ NEXT ERROR RECORD FROM FILE J
210

SUBROUTINE EVAL

This routine evaluates operands which may be either a decimal representation of a constant, a local equate, or a global equate. A test on the first character is performed to see if it is numeric. If it is, then the CONV routine is called which converts it to internal binary form. If it is not a decimal representation of a number, then the local symbol table is searched to see if the operand is a local equate. If it is, the associated symbol table value is returned. If it is not a local equate, then the global symbol table is searched to see if it is a global equate. If it is, then the associated global symbol table value is returned.

SUBROUTINE EVAL
(EVALUATE OPERAND)

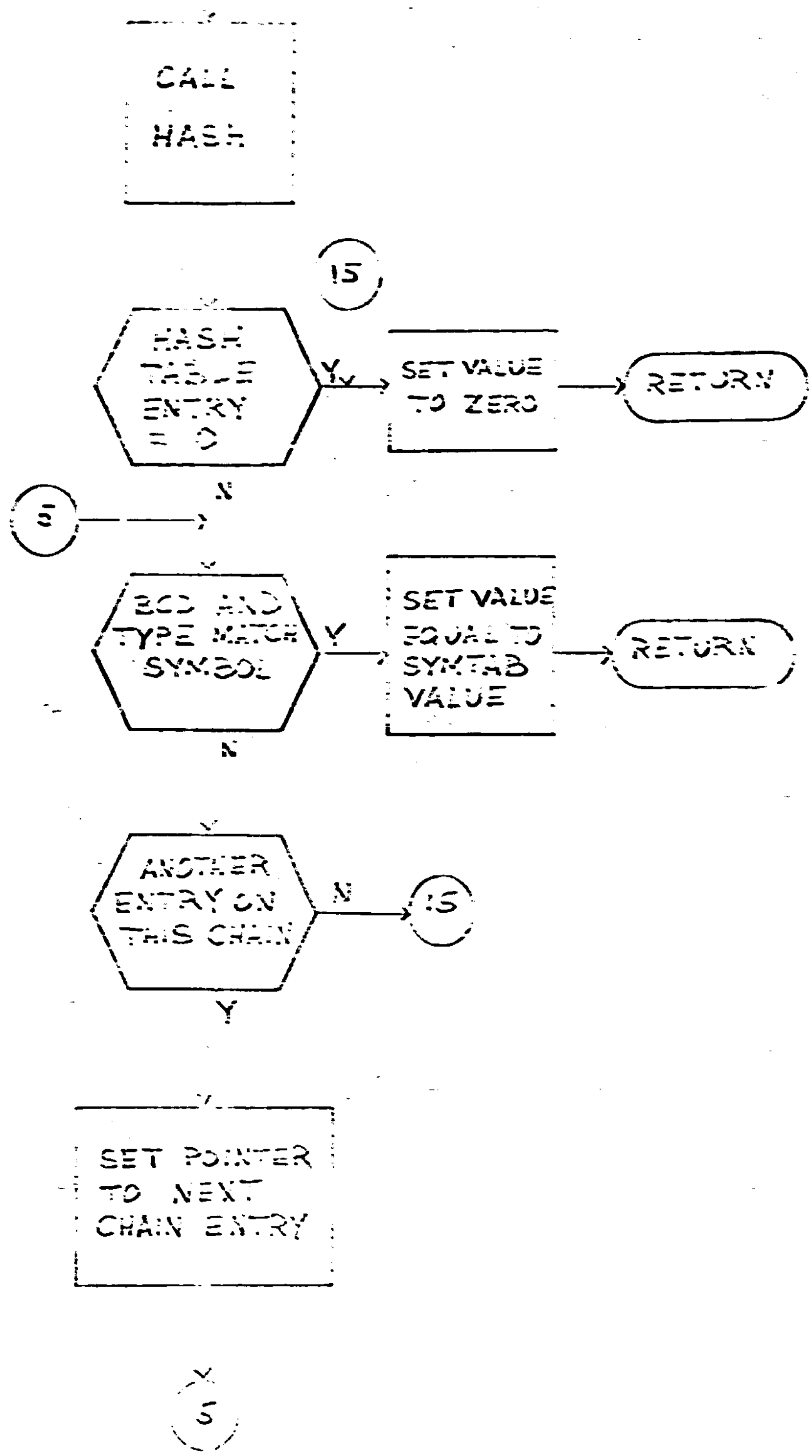



1002
IS
OPERAND
LOCAL
Y
RETURN
A GLOBAL
RETURN
RETURN

SUBROUTINE GGSYM

This routine is used to find the table value for a given global symbol in the global symbol dictionary. The global symbol is hashed into a value from 1 to 100 by calling the HASH routine. This table entry forms the head of a chain through the symbol table entries itself. This chain is searched for the symbol and type of the symbol being searched. When the symbol and type agree, the symbol value is then returned as the output of the subroutine.

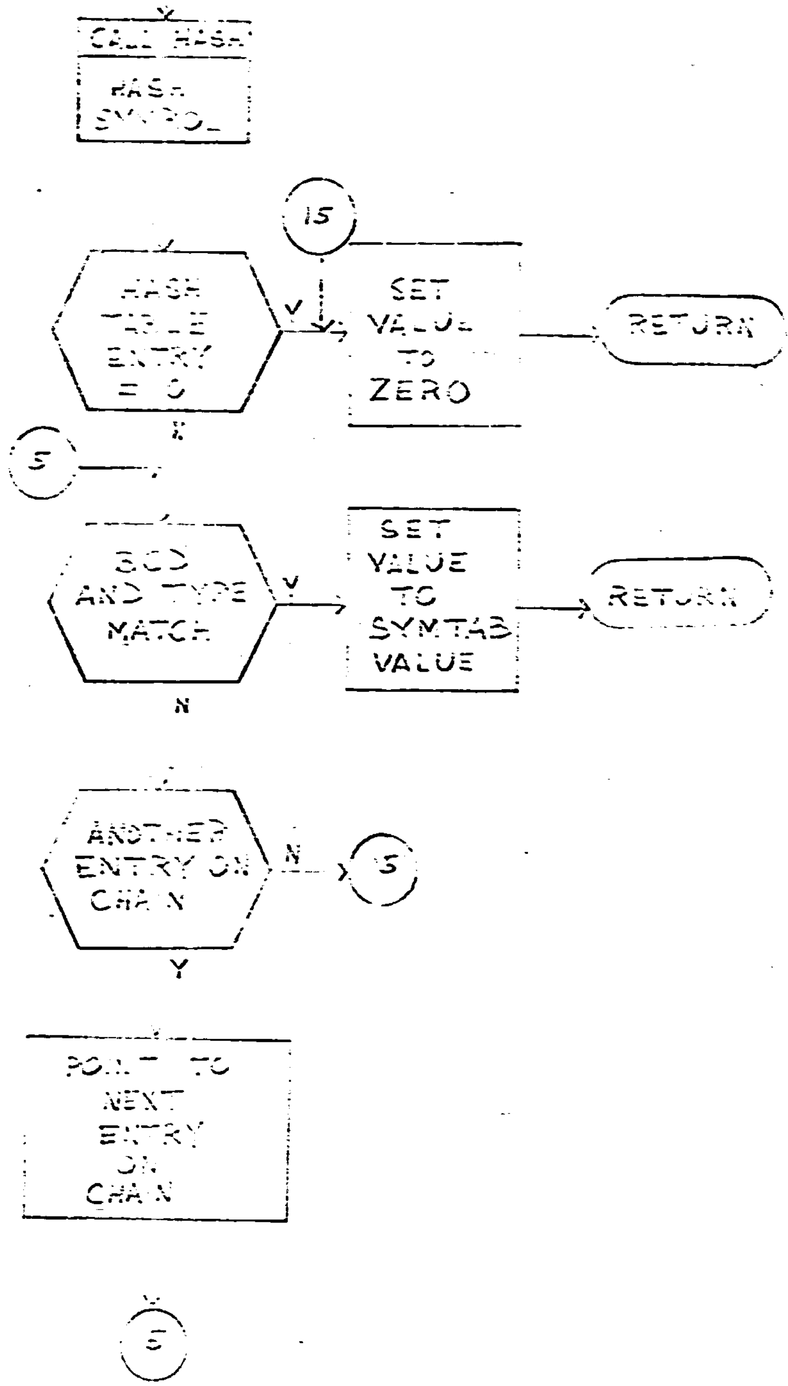
SUBROUTINE GGSYM
(GET FROM GLOBAL SYMTAB)



SUBROUTINE GLSYM

This routine is used to find the table value for a given local symbol in the local symbol dictionary. The local symbol is hashed into a value from 1 to 100 by calling the HASH routine. This table entry forms the head of a chain through the symbol table entries itself. This chain is searched for the symbol and type of the symbol being searched. When the symbol and type agree, the symbol value is then returned as the output of the subroutine.

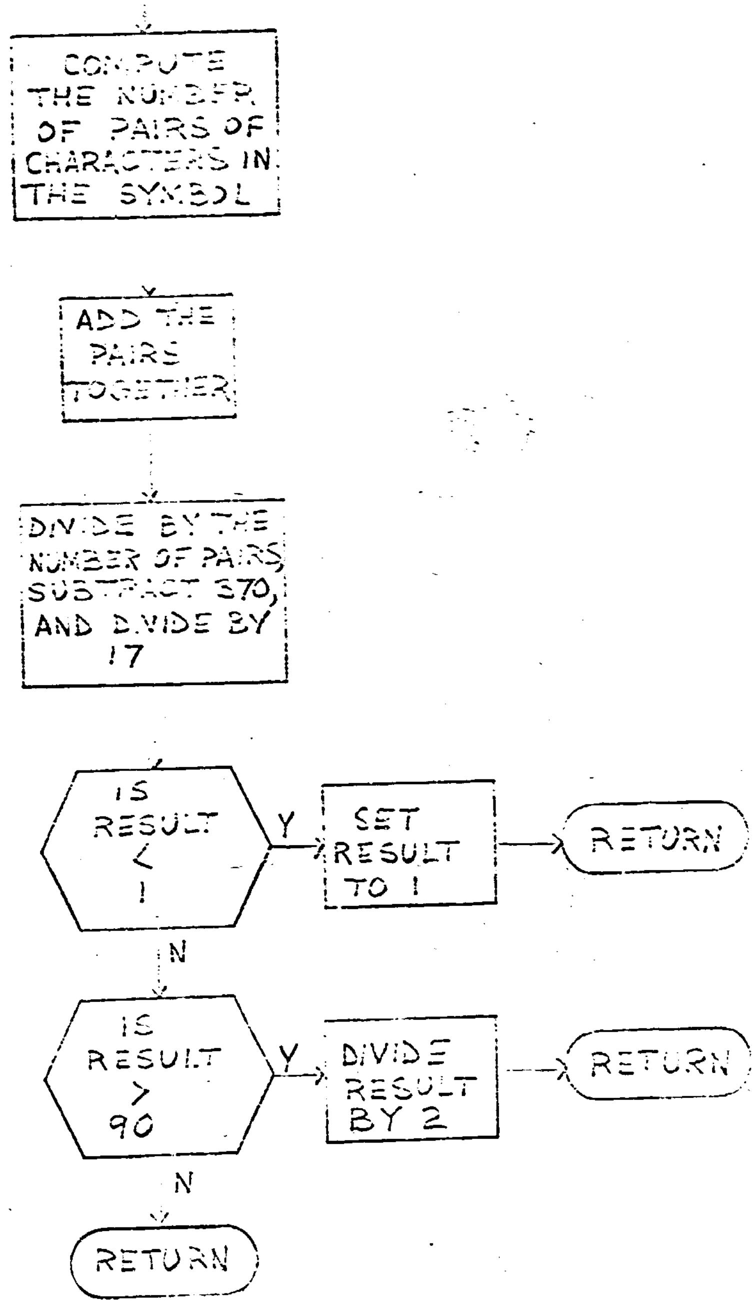
SUBROUTINE GLSYM
(GET FROM LOCAL SYMTAB)



SUBROUTINE HASH

This routine hashes a name in the fix field output area into an integer in the range of 1 to 90. The number of pairs of characters in this symbol are computed. The pairs are then added together and their average value found. A shift and division are performed and a random number between the range of 1 and 90. This number is then returned as the hash value of the symbol.

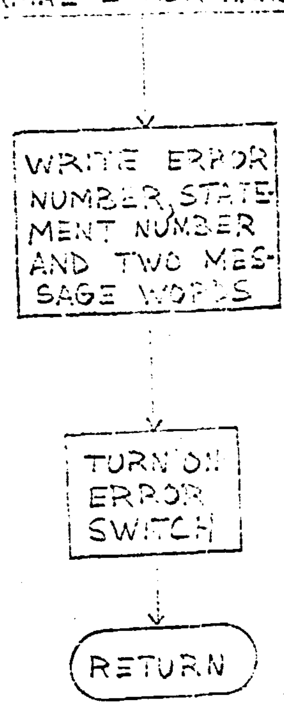
SUBROUTINE HASH
(HASHES SYMBOLS)



SUBROUTINE NERR

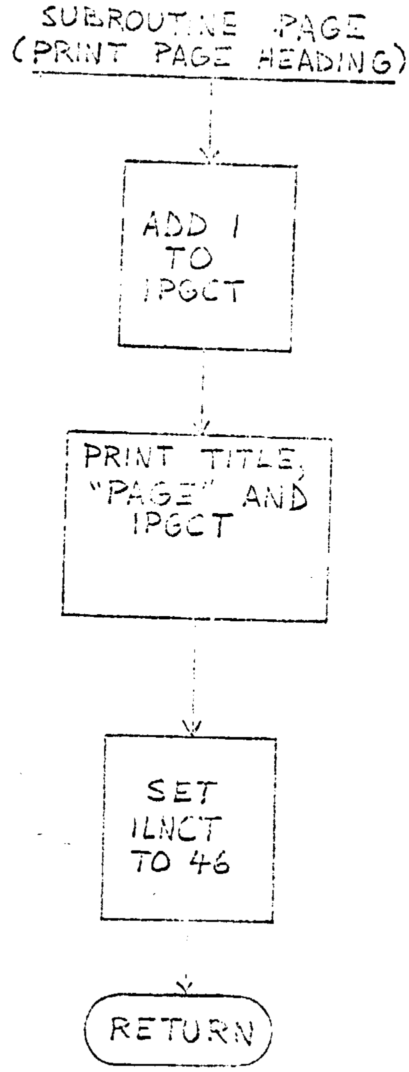
This routine is the normal error handler. It accepts as input an error number, a statement label to which it will return, and two words which are to be included in the error message when it is finally printed out by the error pass. This routine writes the error to the current error file and turns on the master error switch.

SUBROUTINE NERR
(NORMAL ERROR HANDLER)



SUBROUTINE PAGE

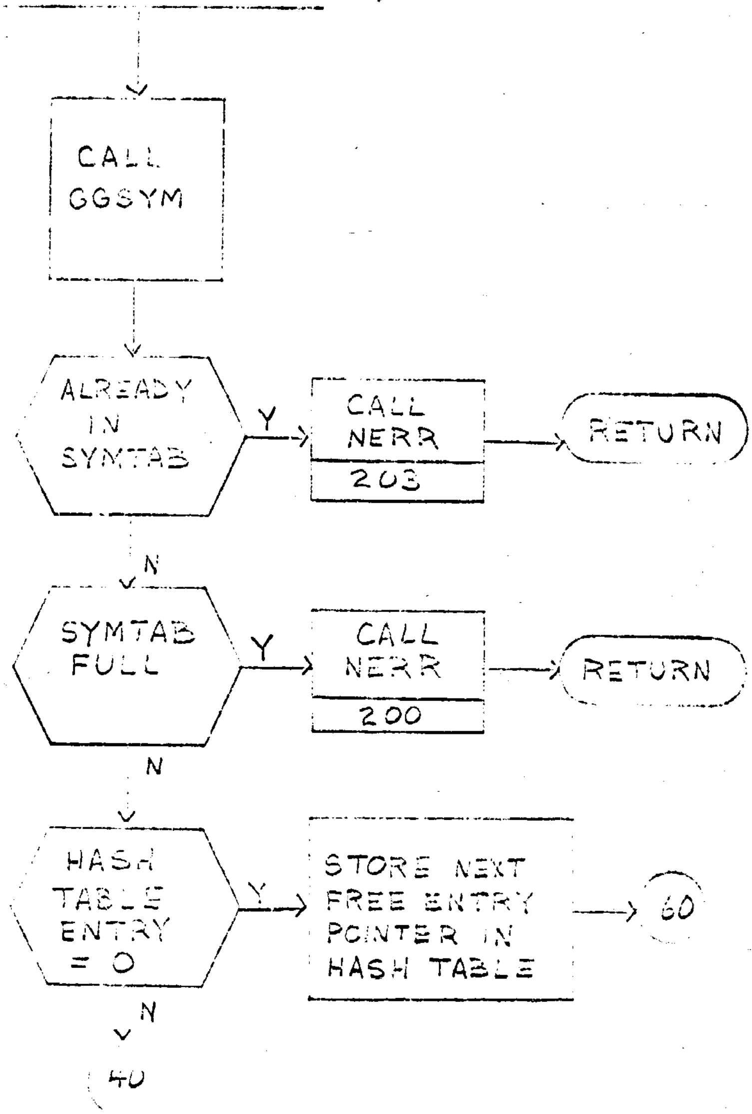
This routine is called whenever it is desired to move to a new page of printout. A title is printed at the top of each page. A count is maintained of the number of pages printed out. This page number also appears on the same line as the title. The routine resets the line counter to a maximum of 46 lines per page.

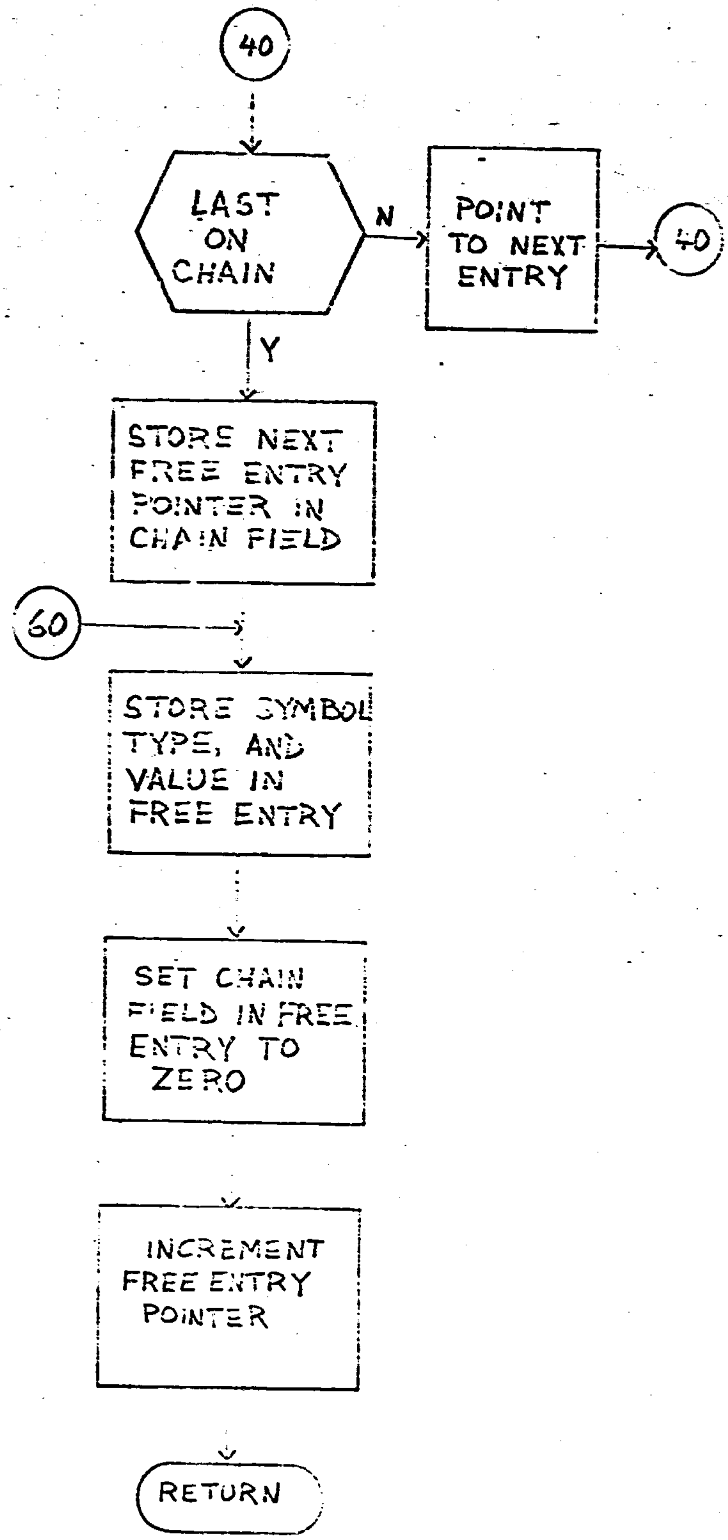

SUBROUTINE PAGE
(PRINT PAGE HEADING)
ADD 1 TO IPGCT
PRINT TITLE, "PAGE" AND IPGCT
SET ILNCT TO 46
RETURN

SUBROUTINE PGSYM

This routine places symbols into the global symbol table. Before placing any symbol into the global symbol table, an attempt is made to see if that symbol is already there, avoiding duplicate entries. If the entry is not in the symbol table, then the hash table entry for that symbol is consulted and the chain followed looking for the last entry on the chain. When the last entry is found, the new symbol is appended to the chain.

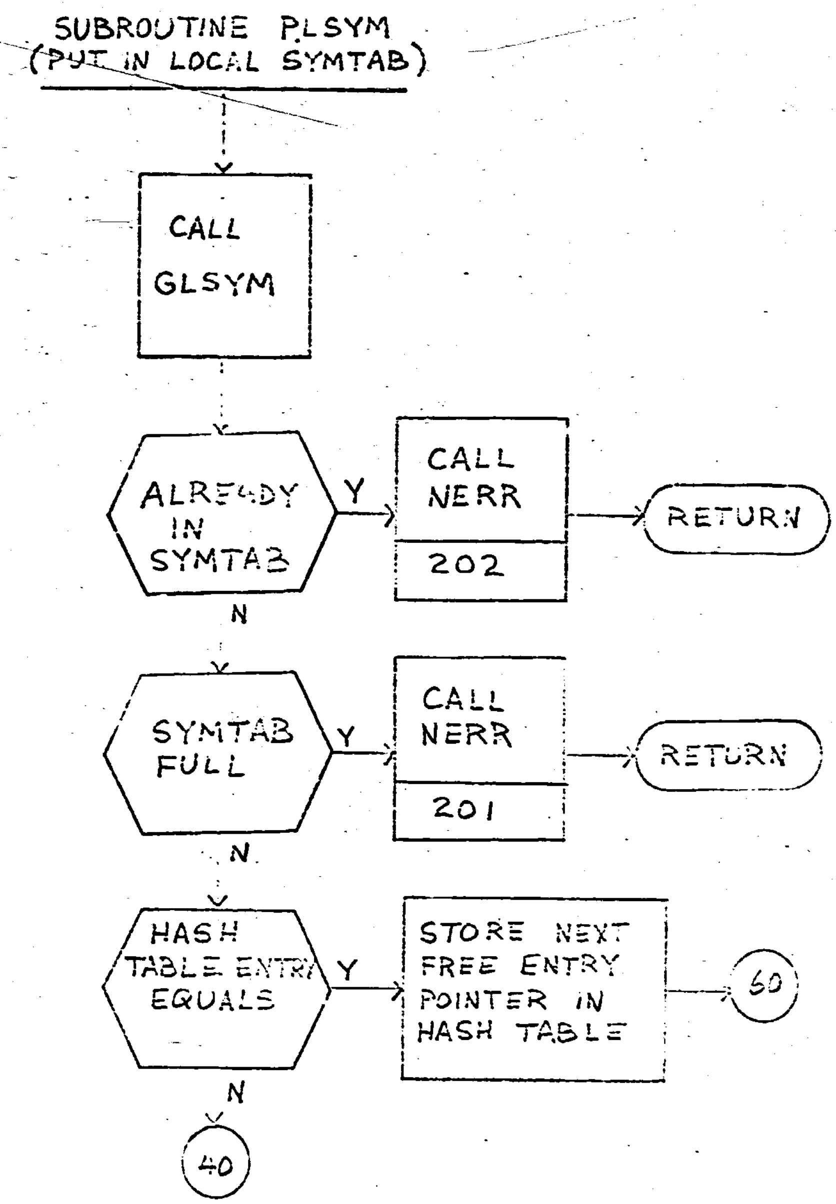
SUBROUTINE PGSYM
(PUT IN GLOBAL SYMTAB)

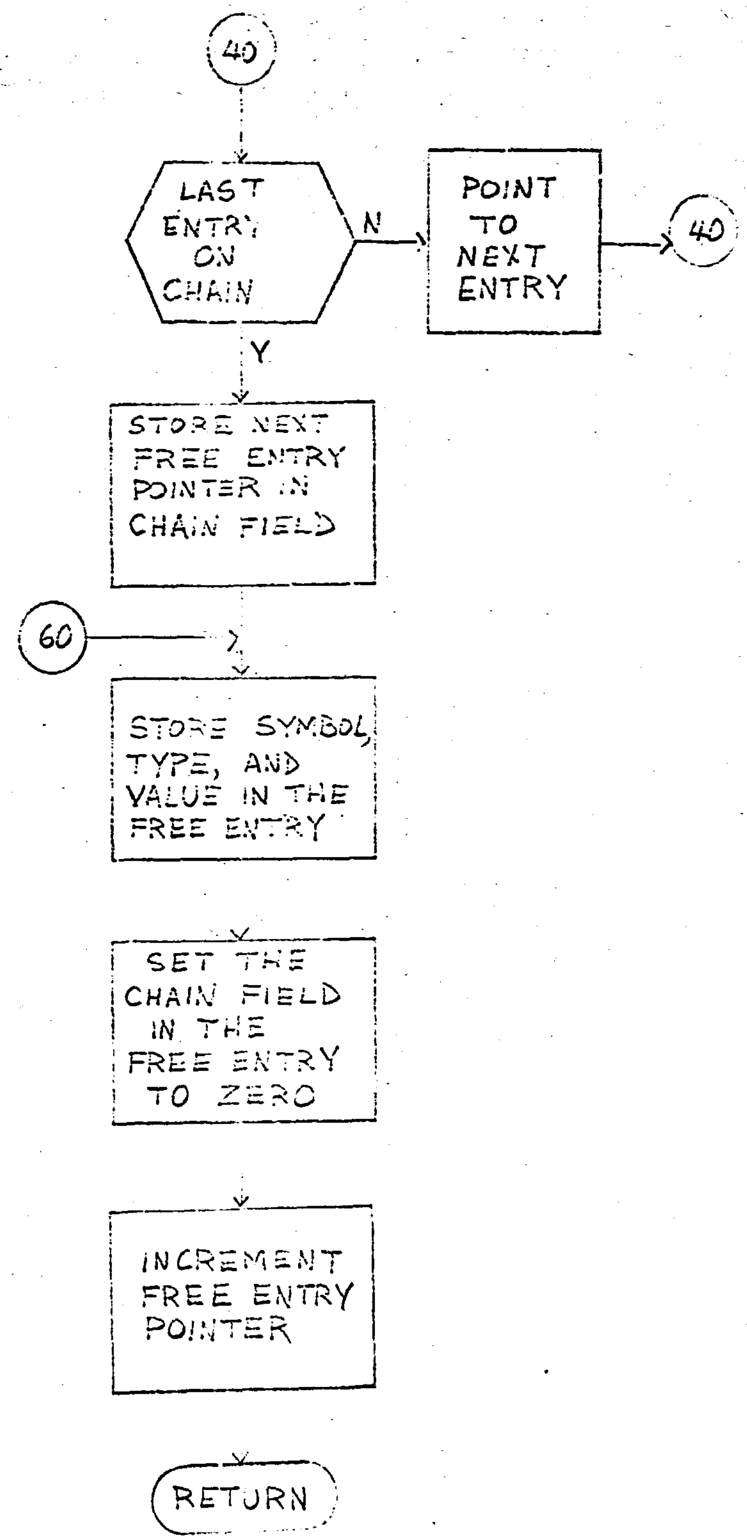



40
LAST ON CHAIN
N
POINT TO NEXT ENTRY
40
Y
STORE NEXT FREE ENTRY POINTER IN CHAIN FIELD
60
STORE SYMBOL TYPE, AND VALUE IN FREE ENTRY
SET CHAIN FIELD IN FREE ENTRY TO ZERO
INCREMENT FREE ENTRY POINTER
RETURN

SUBROUTINE PLSYM

This routine places symbols into the local symbol table. Before placing any symbol into the local symbol table, an attempt is made to see if that symbol is already there, avoiding duplicate entries. If the entry is not in the symbol table, then the hash table entry for that symbol is consulted and the chain followed looking for the last entry on the chain. When the last entry is found, the new symbol is appended to the chain.

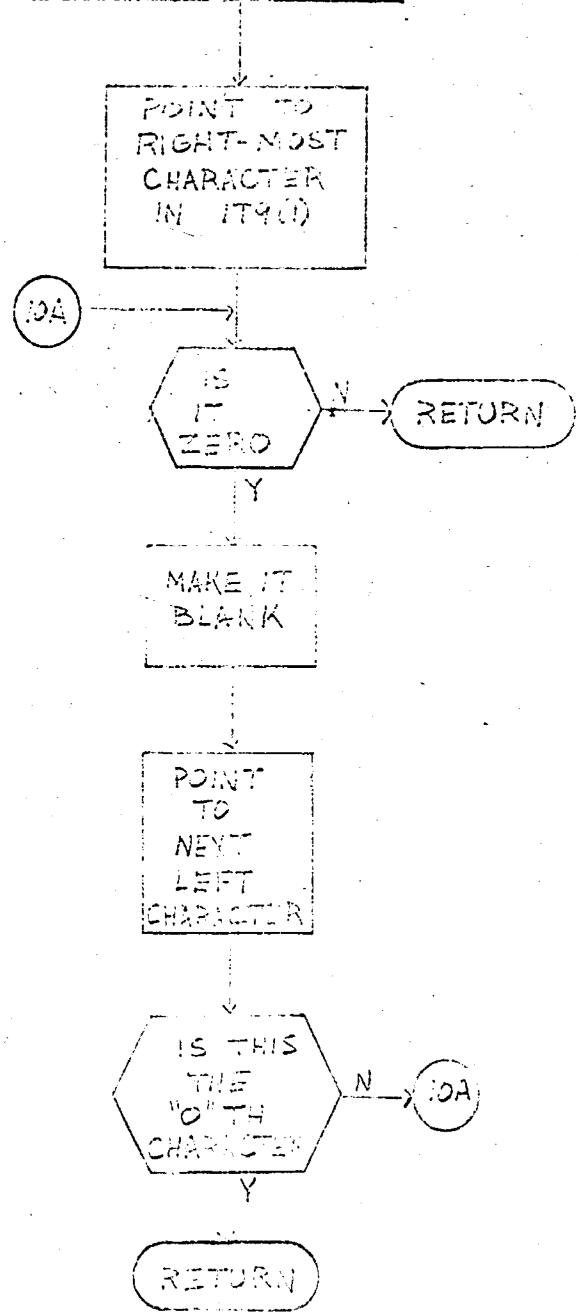

SUBROUTINE PLSYM
(PUT IN LOCAL SYMTAB)
CALL GLSYM
ALREADY IN SYMTAB
Y
CALL NERR
202
RETURN
N
SYMTAB FULL
Y
CALL NERR
201
RETURN
N
HASH TABLE ENTRY EQUALS
Y
STORE NEXT FREE ENTRY POINTER IN HASH TABLE
60
N
40


40
LAST ENTRY ON CHAIN
N
POINT TO NEXT ENTRY
40
Y
STORE NEXT FREE ENTRY POINTER IN CHAIN FIELD
60
STORE SYMBOL, TYPE, AND VALUE IN THE FREE ENTRY
SET THE CHAIN FIELD IN THE FREE ENTRY TO ZERO
INCREMENT FREE ENTRY POINTER
RETURN

SUBROUTINE RBLK

This routine changes right hand zeroes to blanks in the first word of the input area used to read from the library. This padding with blanks is performed prior to searching the library for the given entry point.

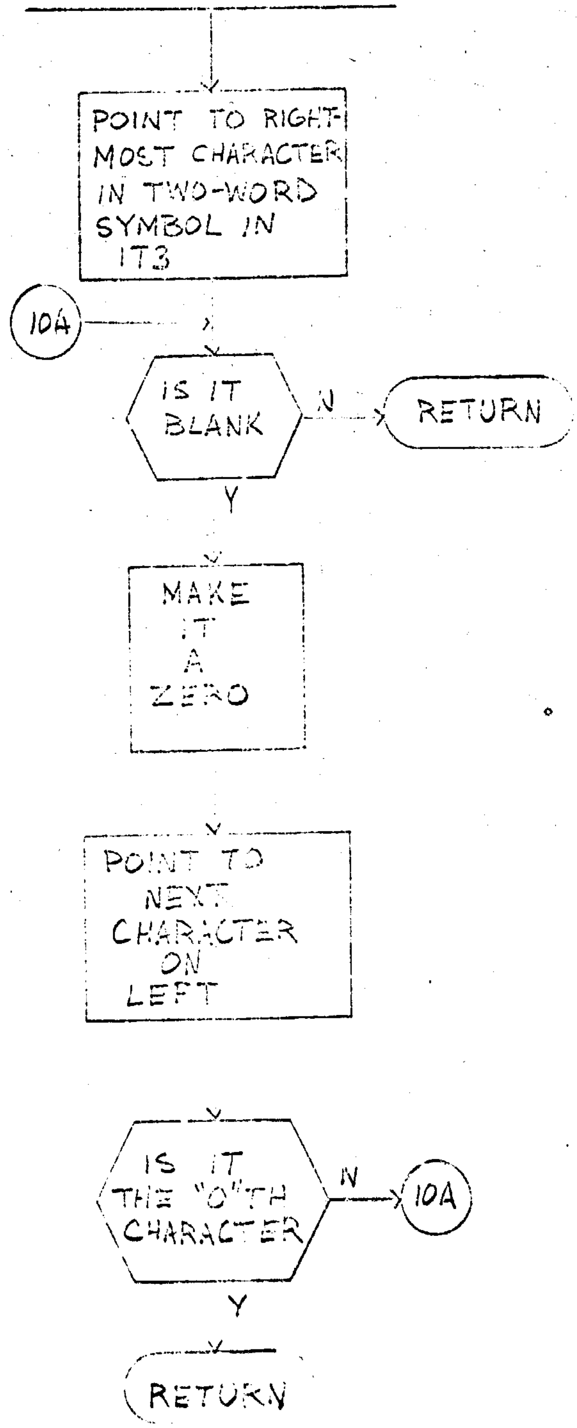
SUBROUTINE RBLK
(RIGHT FILL WITH BLANKS)



SUBROUTINE RZRO

This routine is used to right-fill a two word symbol in the fix field output area with zeroes.

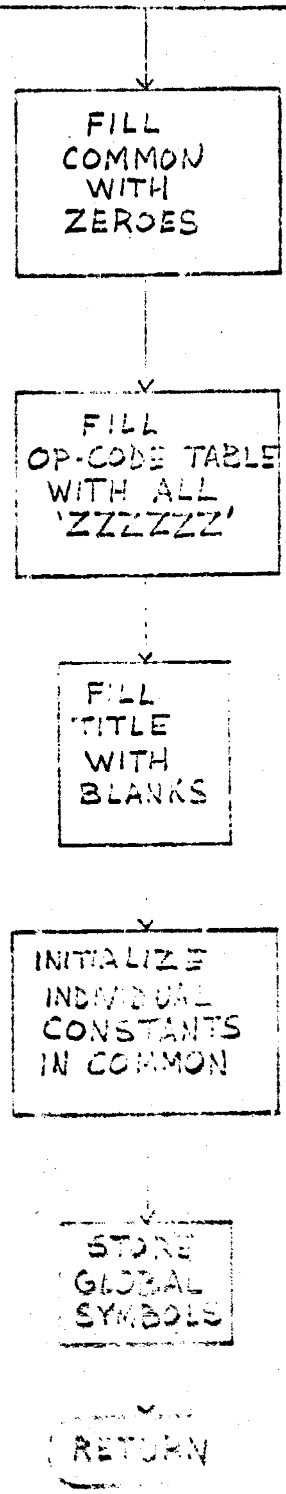
SUBROUTINE RZRO
(RIGHT FILL WITH ZEROS)



SUBROUTINE SA

This subroutine initializes phase 2 by filling common with zeroes, filling the op-code table with all 'Z's, filling the title with blanks, setting constants, and storing global symbols.

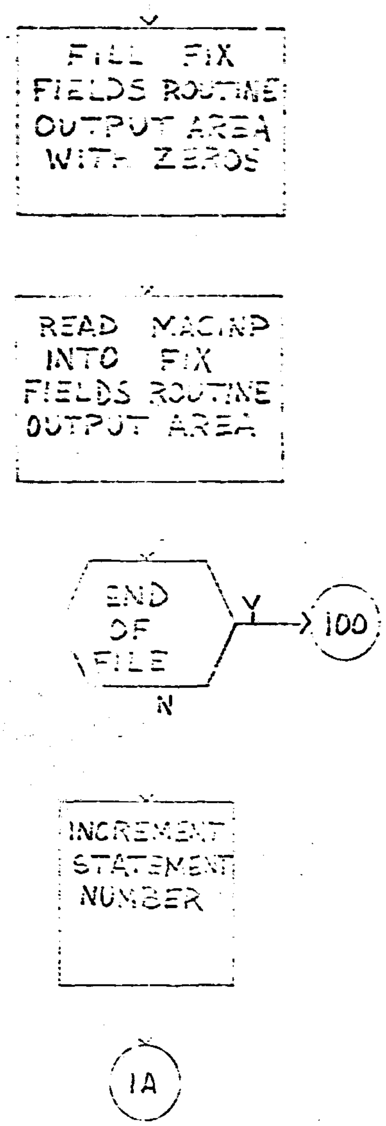
SUBROUTINE SA
(INITIALIZE PHASE 2)

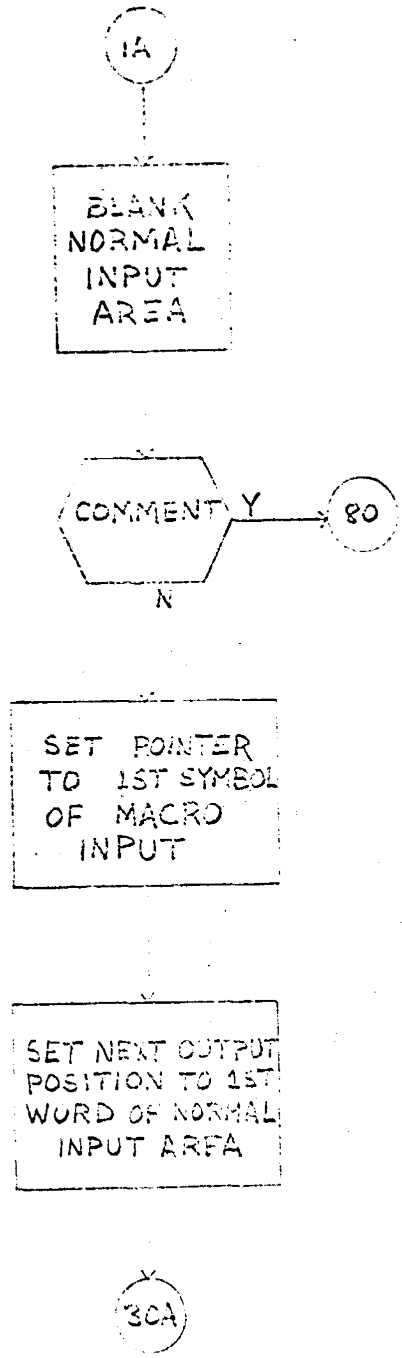


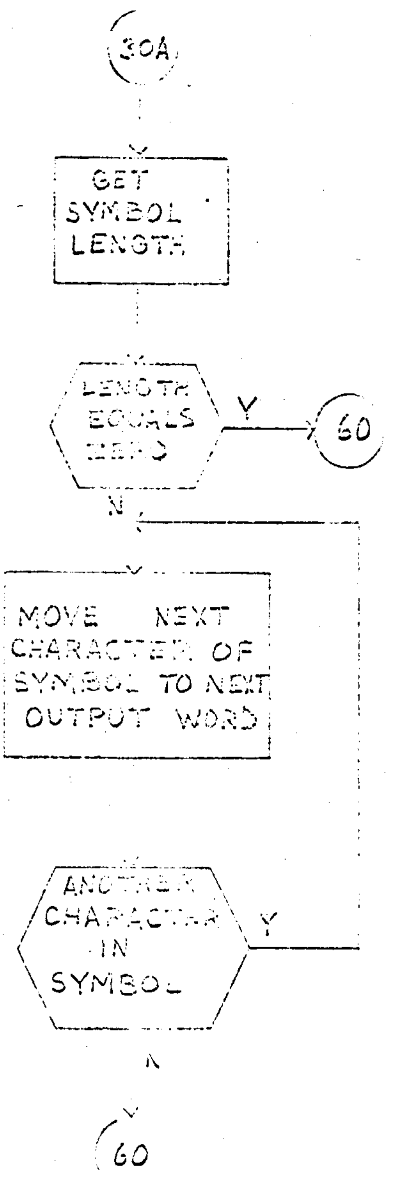
SUBROUTINE SA1

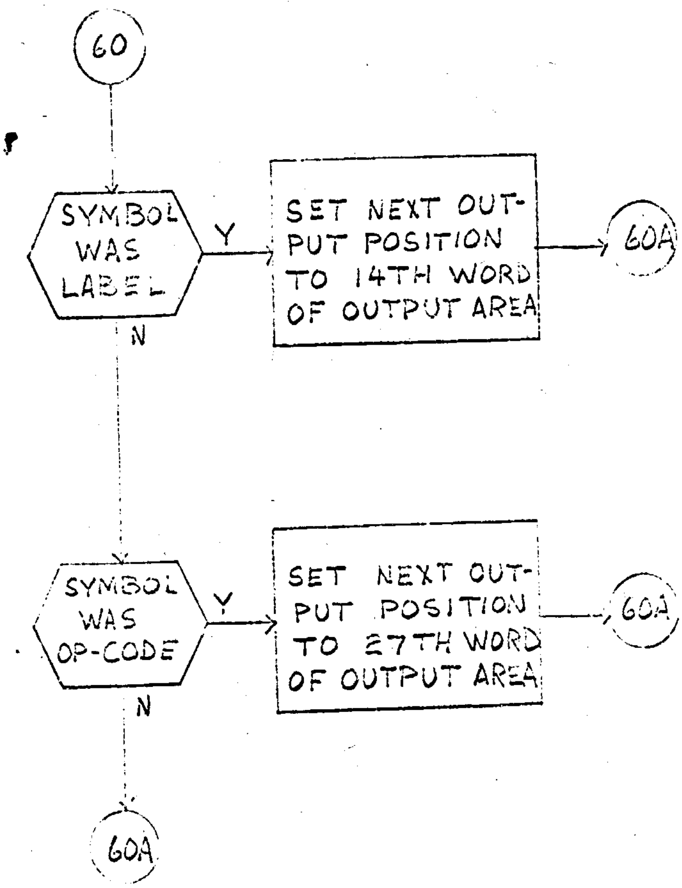
This routine reads input from the macro input file in fixed fields format. It also reconstructs the statement in preparation for printing out. Upon reaching the end of the macro input file it sets an end of file indicator.

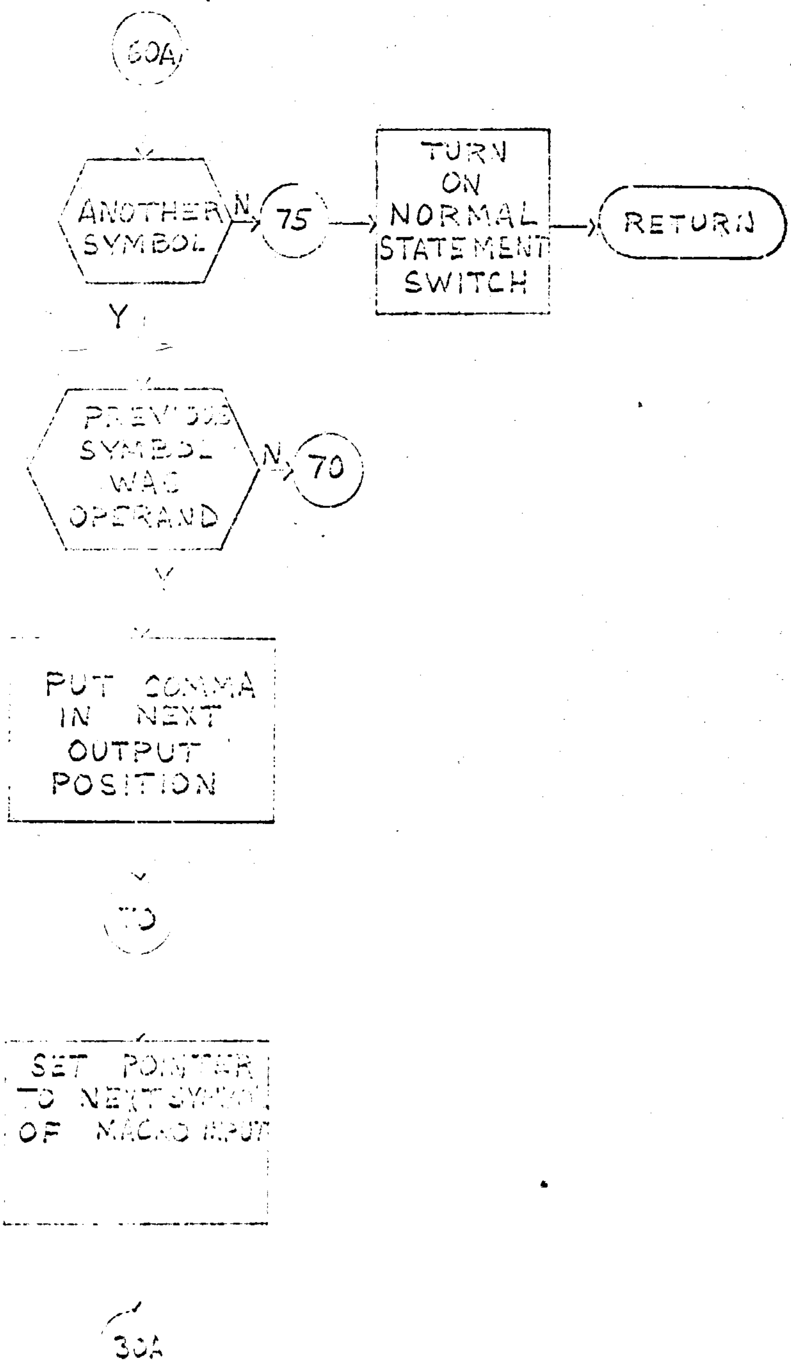
SUBROUTINE SA1
(READ MACRO INPUT)

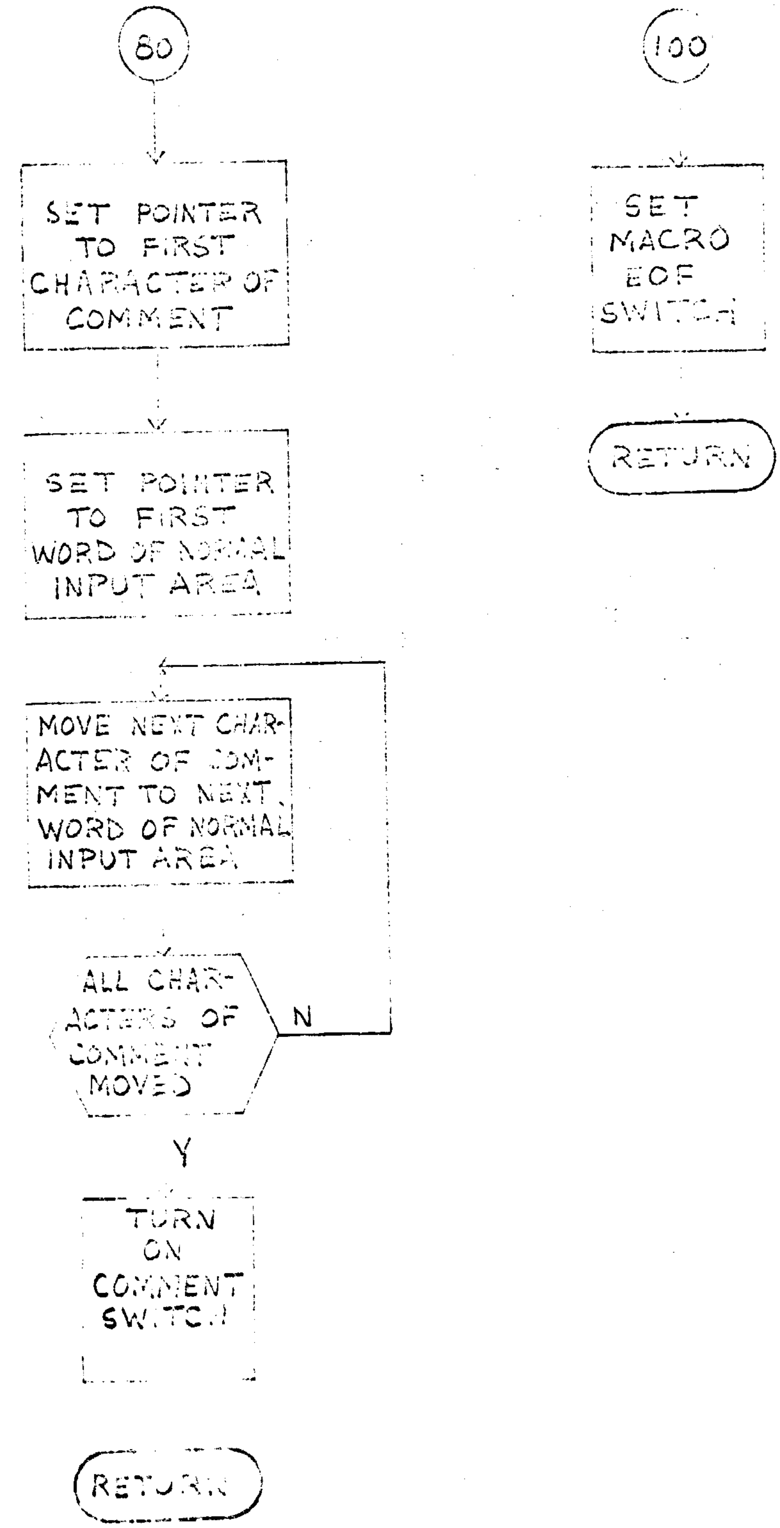



1A
BLANK NORMAL INPUT AREA
COMMENT
Y
80
N
SET POINTER TO 1ST SYMBOL OF MACRO INPUT
SET NEXT OUTPUT POSITION TO 1ST WORD OF NORMAL INPUT AREA
3CA


30A
GET SYMBOL LENGTH
LENGTH EQUALS ZERO
Y
60
N
MOVE NEXT CHARACTER OF SYMBOL TO NEXT OUTPUT WORD
ANOTHER CHARACTER IN SYMBOL
Y
N
60


60
SYMBOL WAS LABEL
Y
N
SET NEXT OUT-PUT POSITION TO 14TH WORD OF OUTPUT AREA
60A
SYMBOL WAS OP-CODE
Y
N
SET NEXT OUT-PUT POSITION TO 27TH WORD OF OUTPUT AREA
60A
60A

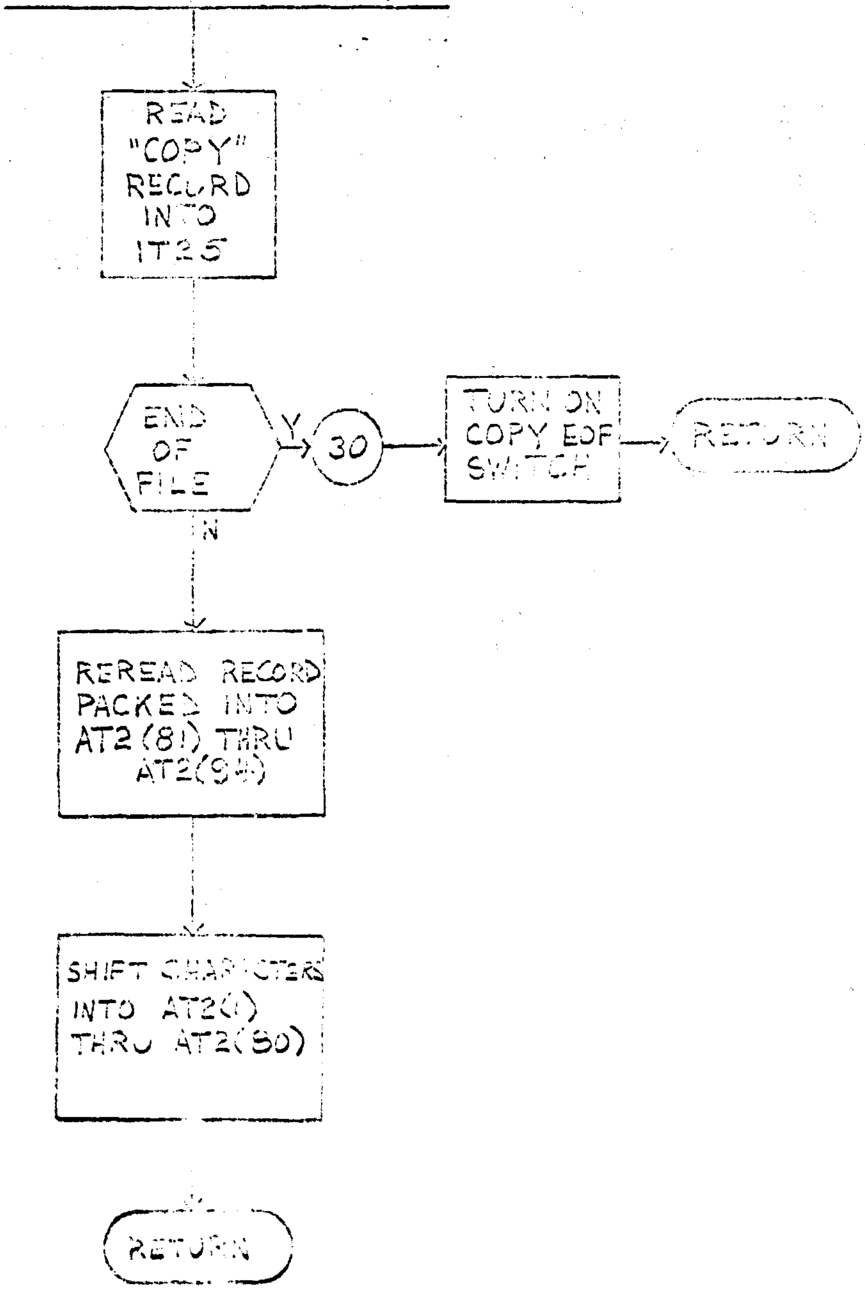

30A
ANOTHER SYMBOL
N
75
TURN ON NORMAL STATEMENT SWITCH
RETURN
Y
PREVIOUS SYMBOL WAS OPERAND
N
70
Y
PUT COMMA IN NEXT OUTPUT POSITION
70
SET POINTER TO NEXT SYMBOL OF MACRO INPUT
30A


80
SET POINTER TO FIRST CHARACTER OF COMMENT
SET POINTER TO FIRST WORD OF NORMAL INPUT AREA
MOVE NEXT CHARACTER OF COMMENT TO NEXT WORD OF NORMAL INPUT AREA
ALL CHARACTERS OF COMMENT MOVED
N
Y
TURN ON COMMENT SWITCH
RETURN
100
SET MACRO EOF SWITCH
RETURN

SUBROUTINE SA2

This routine reads source statements that have been placed on the copy input file following execution of a copy statement. Each call on this subroutine causes one record to be read from the copy input file. At the end of the file, an end of file switch is set.

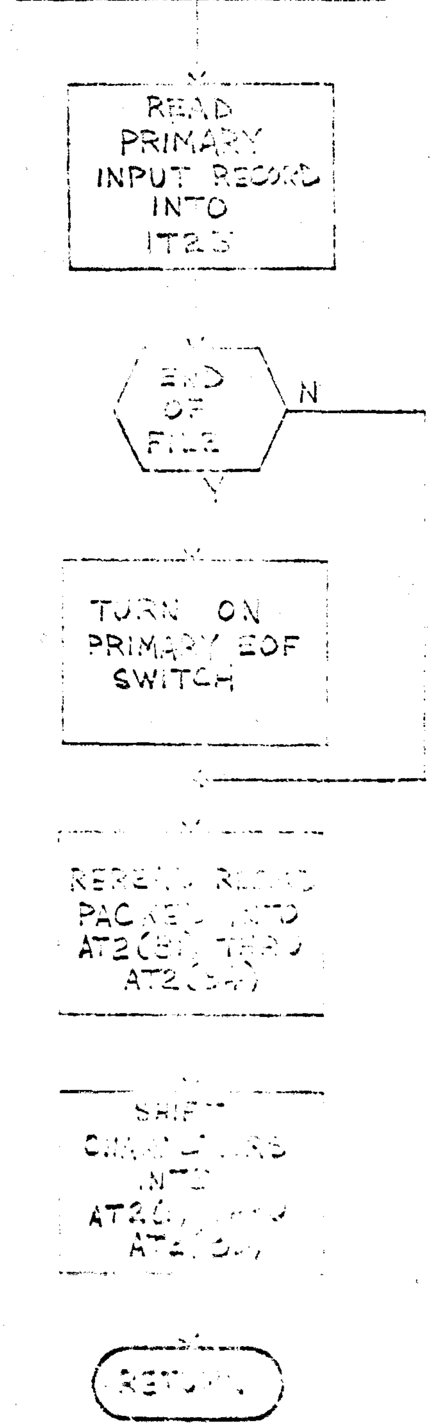
SUBROUTINE SA2
(READ COPY-INPUT)



SUBROUTINE SA3

This routine reads source statements from the standard input file. Each call on this routine causes one record to be read from the standard input file. The input is stored in both packed and unpacked forms for further processing. When the end of file is reached, an end of file switch is set.

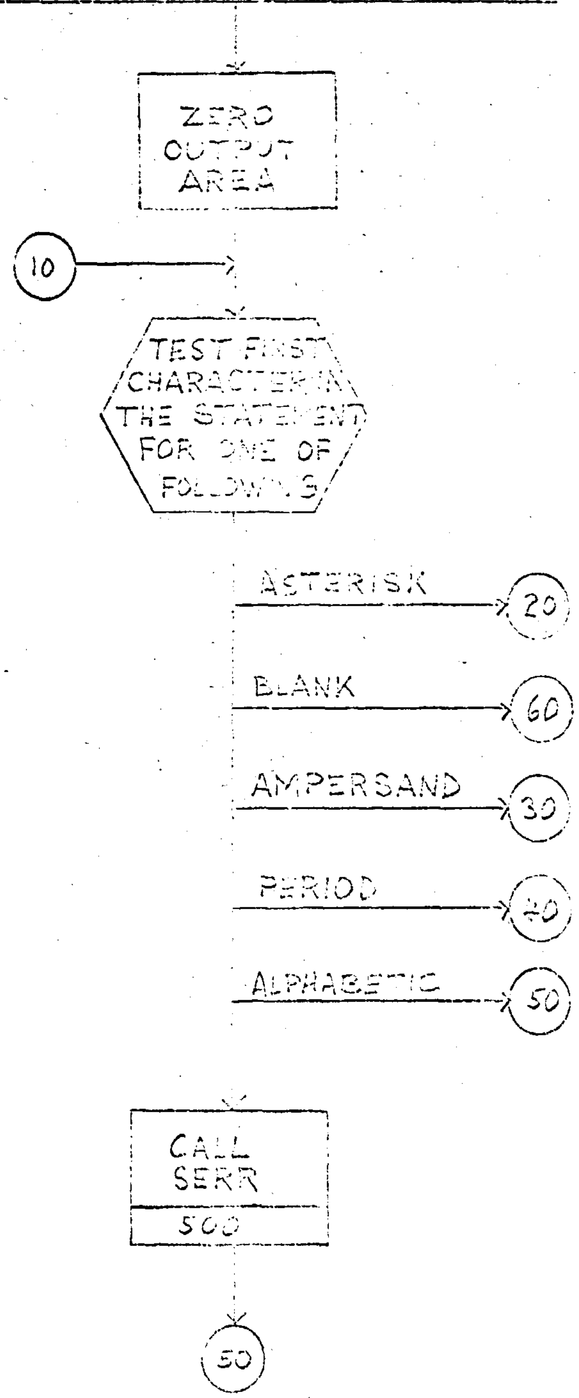
SUBROUTINE SA3
(READ PRIMARY INPUT)

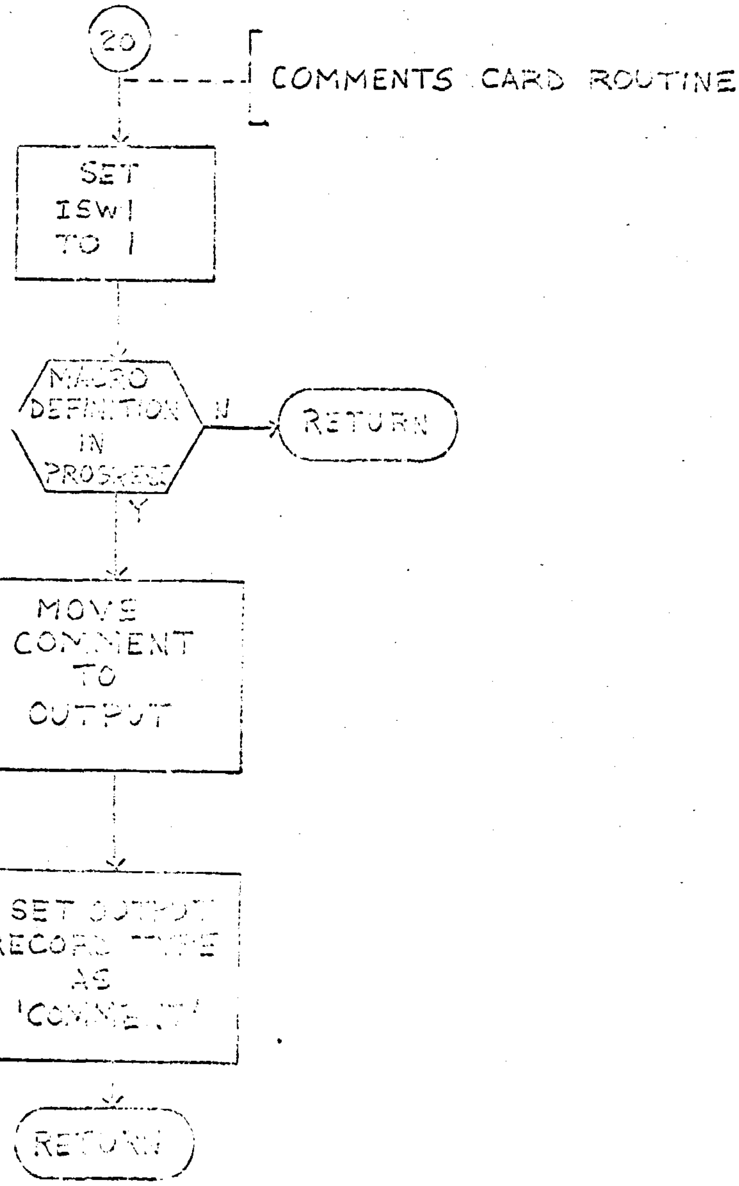


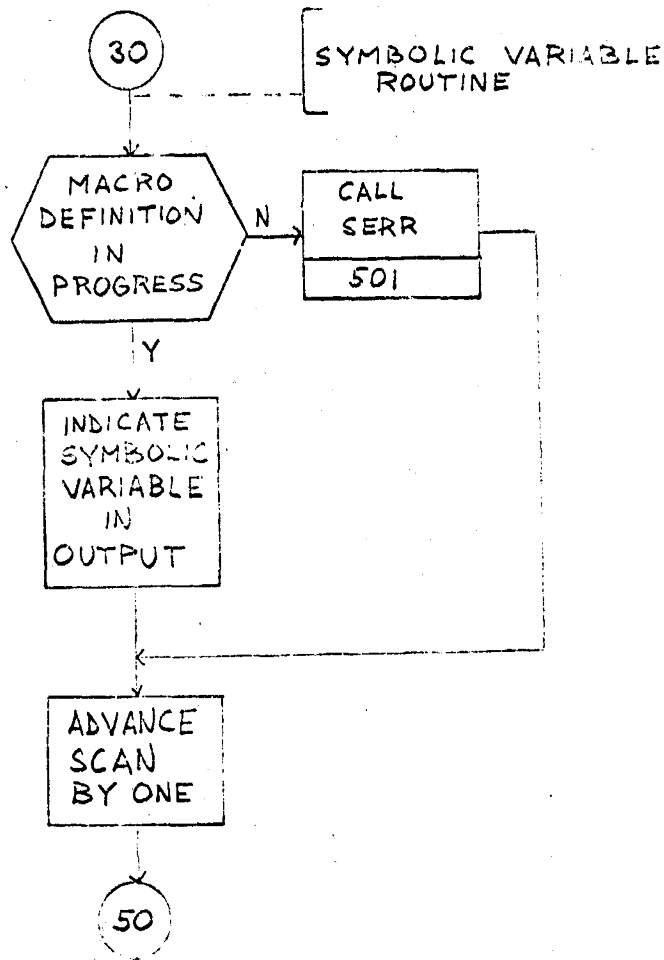
SUBROUTINE SA4

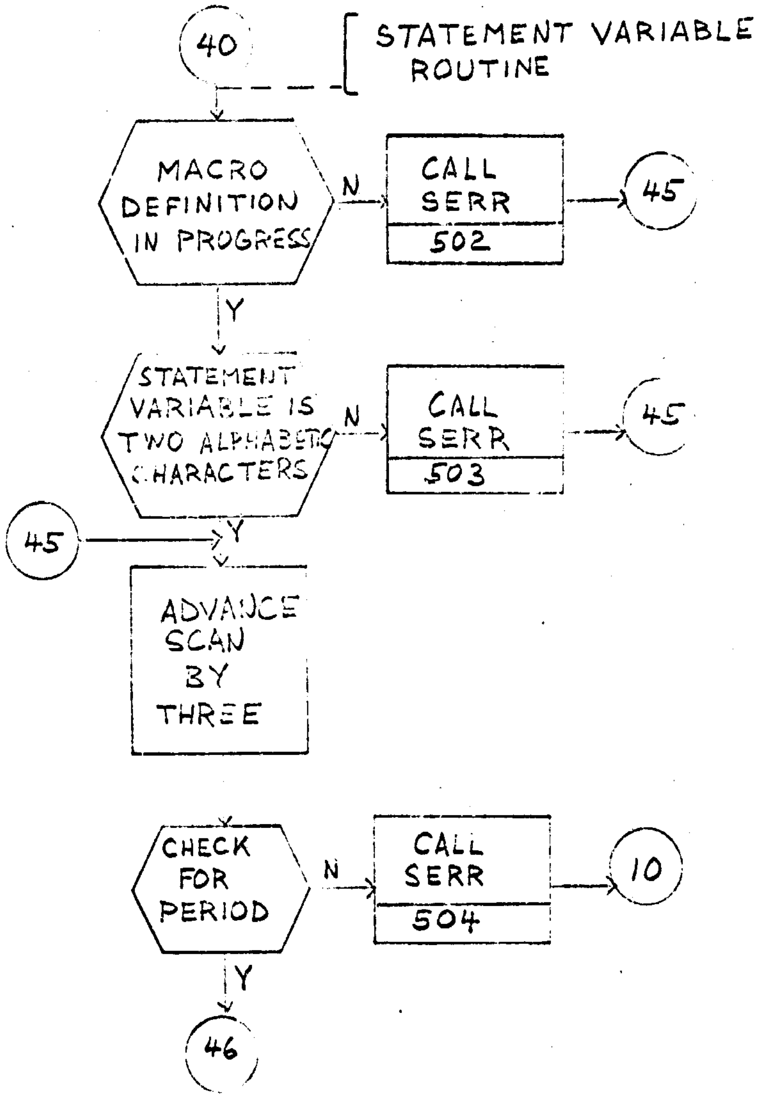
This routine transforms the free field format input into a fixed field form used for processing by all other routines in the first pass. Within this subroutine, there are separate routines for processing comment cards, macro definitions, special op-codes, normal op-codes, continuation cards, symbols followed by asterisks, symbols preceded by ampersands, and statement label variables. This routine has a secondary entry point, SA5, which is called whenever a continuation card is to be transformed into fixed field output form.

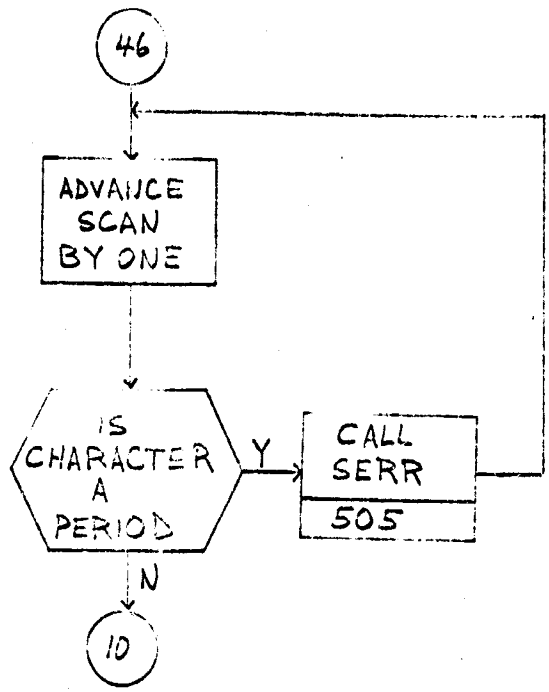
SUBROUTINE SA4
(FIX FIELDS ROUTINE)

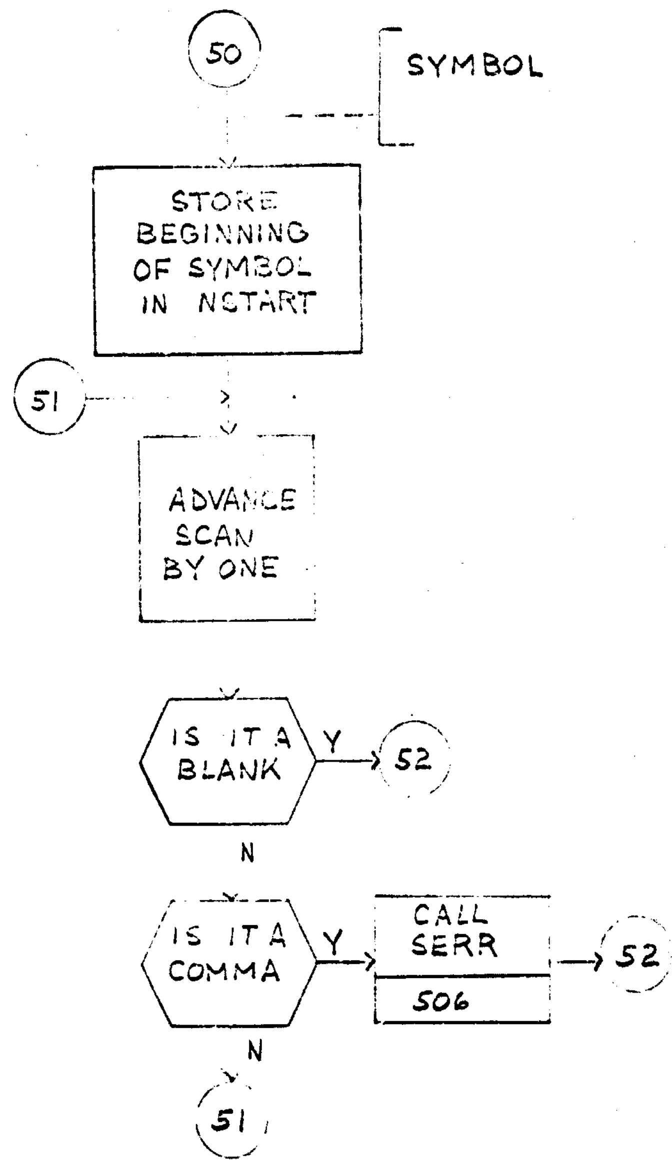


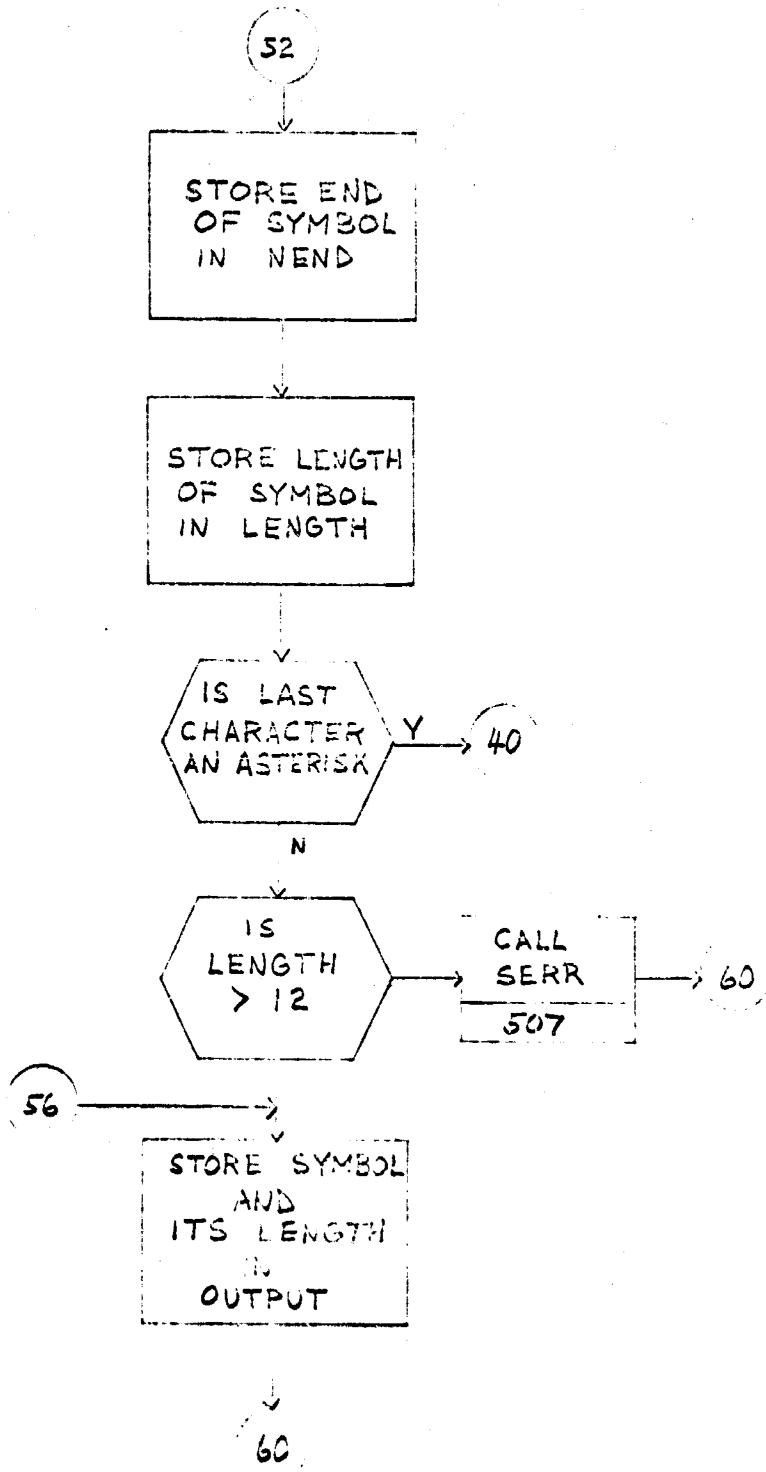

20
COMMENTS CARD ROUTINE
SET ISW1 TO 1
MACRO DEFINITION IN PROGRESS
N
RETURN
Y
MOVE COMMENT TO OUTPUT
SET OUTPUT RECORD TYPE AS 'COMMENT'
RETURN

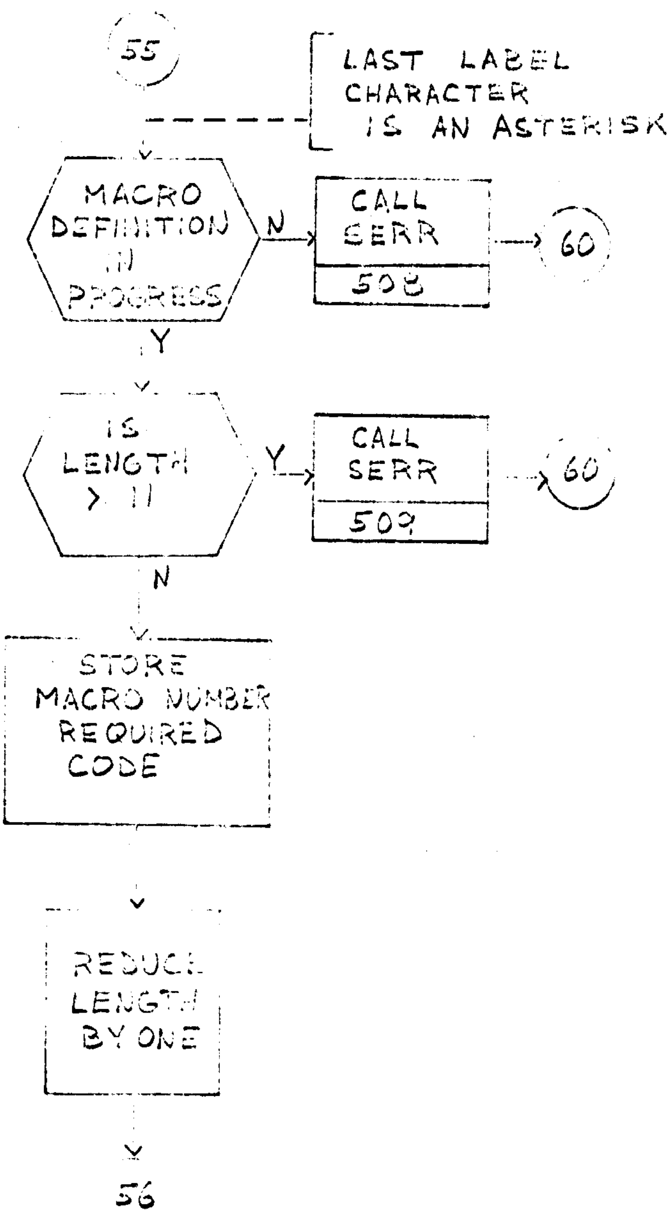

30
SYMBOLIC VARIABLE ROUTINE
MACRO DEFINITION IN PROGRESS
N
CALL SERR
501
Y
INDICATE SYMBOLIC VARIABLE IN OUTPUT
ADVANCE SCAN BY ONE
50

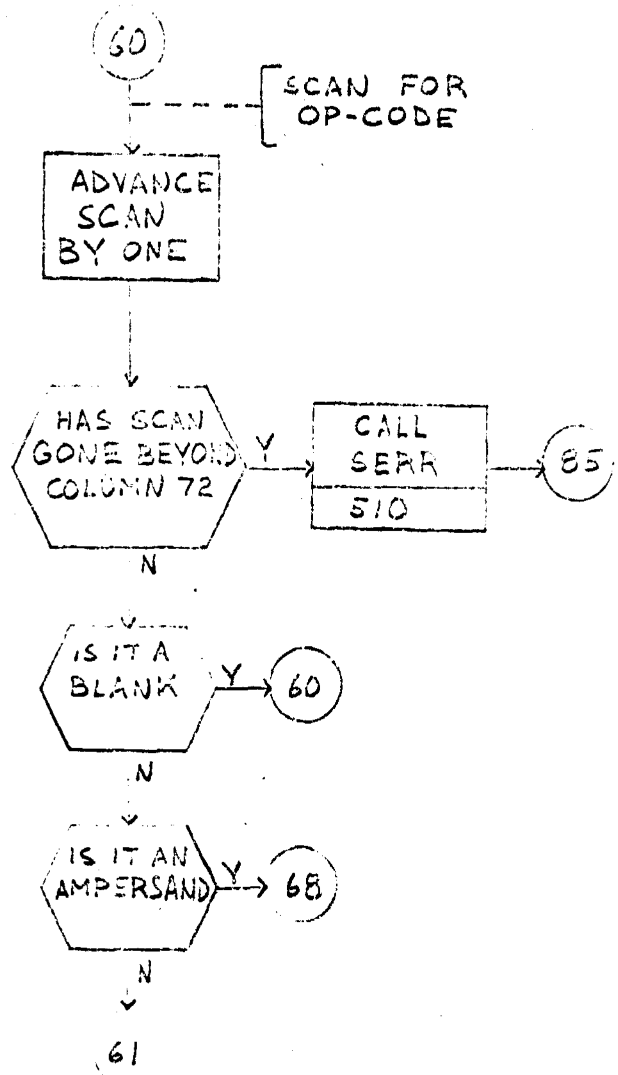

40
STATEMENT VARIABLE ROUTINE
MACRO DEFINITION IN PROGRESS
N
CALL SERR
502
45
Y
STATEMENT VARIABLE IS TWO ALPHABETIC CHARACTERS
N
CALL SERR
503
45
45
Y
ADVANCE SCAN BY THREE
CHECK FOR PERIOD
N
CALL SERR
504
10
Y
46

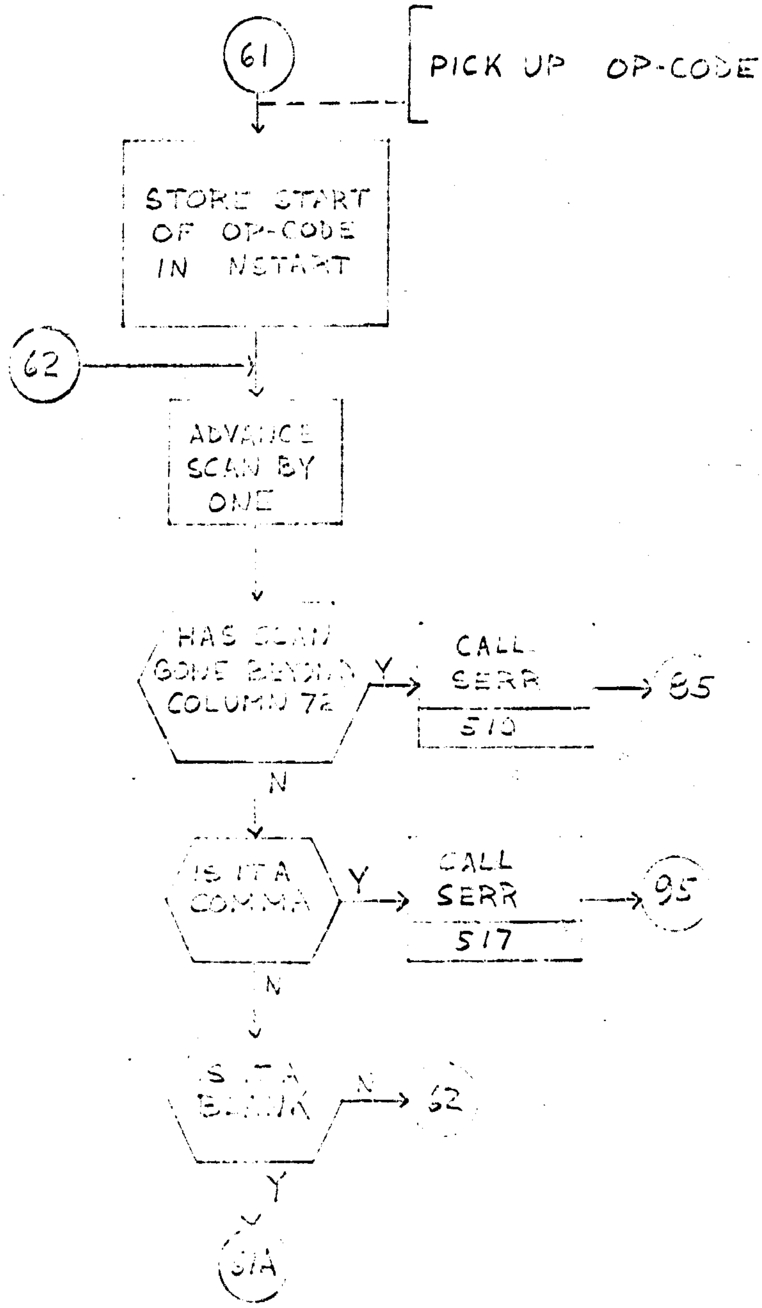

46
ADVANCE SCAN BY ONE
IS CHARACTER A PERIOD
Y
N
CALL SERR
505
10

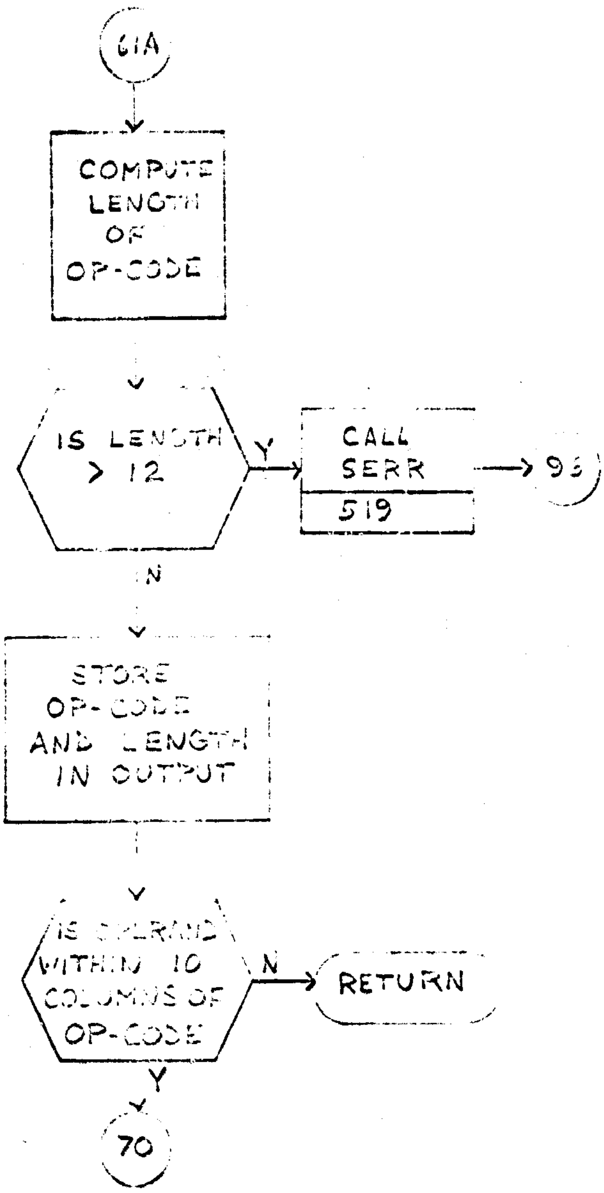

50
SYMBOL
STORE BEGINNING OF SYMBOL IN NSTART
51
ADVANCE SCAN BY ONE
IS IT A BLANK
Y
52
N
IS IT A COMMA
Y
CALL SERR
506
52
N
51

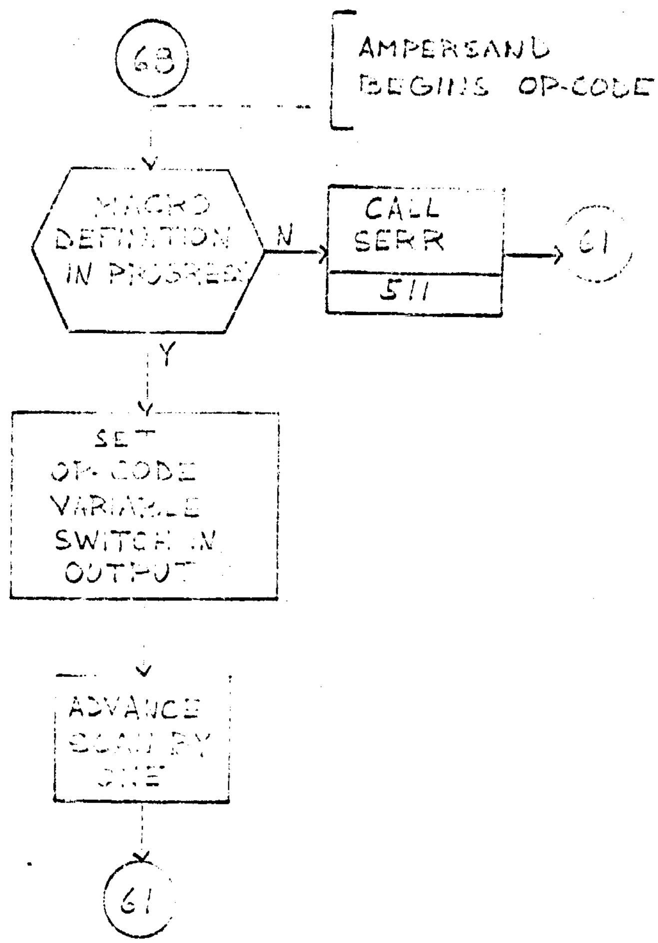

52
STORE END OF SYMBOL IN NEND
STORE LENGTH OF SYMBOL IN LENGTH
IS LAST CHARACTER AN ASTERISK
Y
40
N
IS LENGTH > 12
CALL SERR
507
60
56
STORE SYMBOL AND ITS LENGTH IN OUTPUT
60

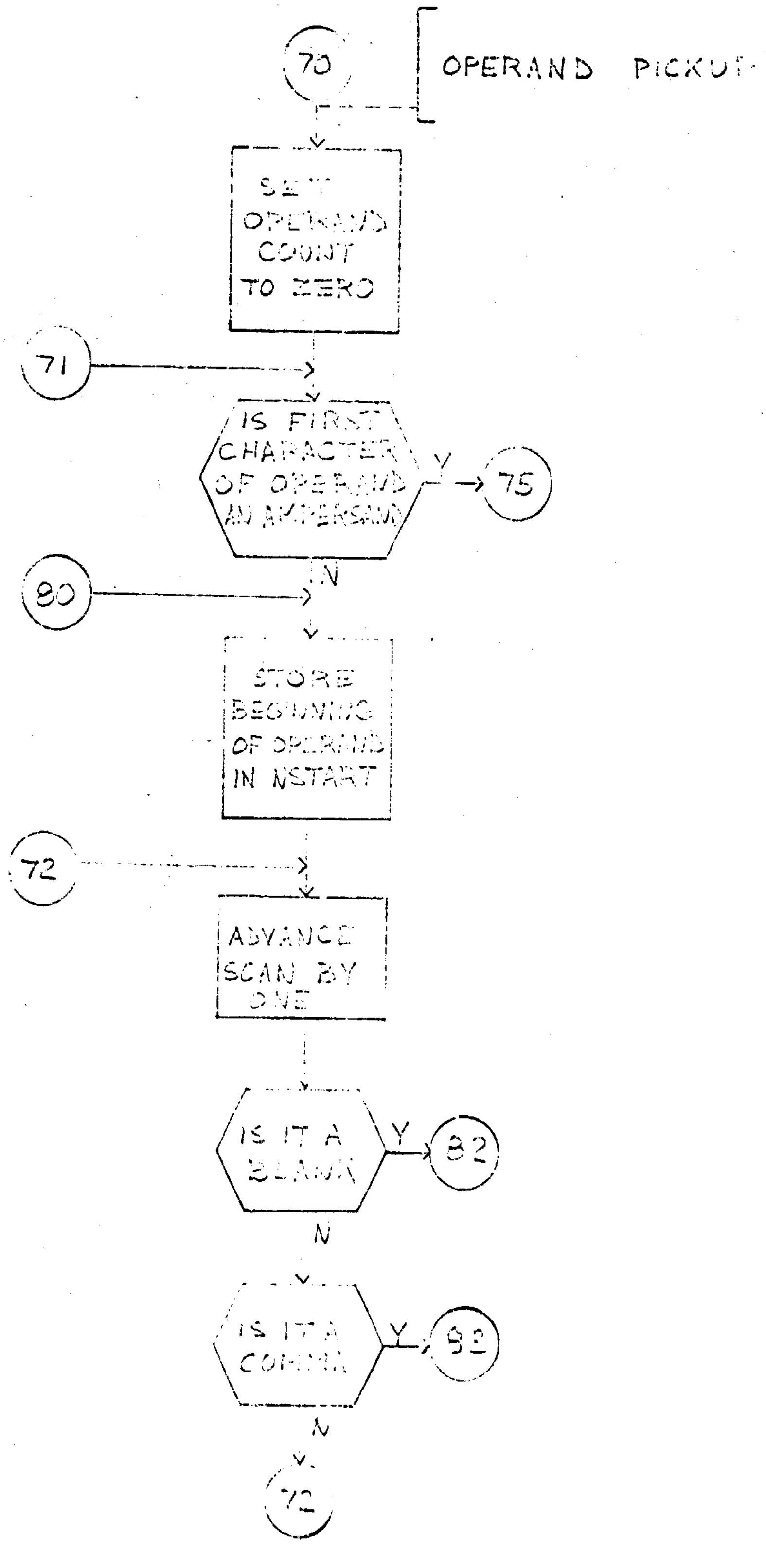

55
LAST LABEL CHARACTER IS AN ASTERISK
MACRO DEFINITION IN PROGRESS
N
CALL SERR
508
60
Y
IS LENGTH > 11
Y
CALL SERR
509
60
N
STORE MACRO NUMBER REQUIRED CODE
REDUCE LENGTH BY ONE
56

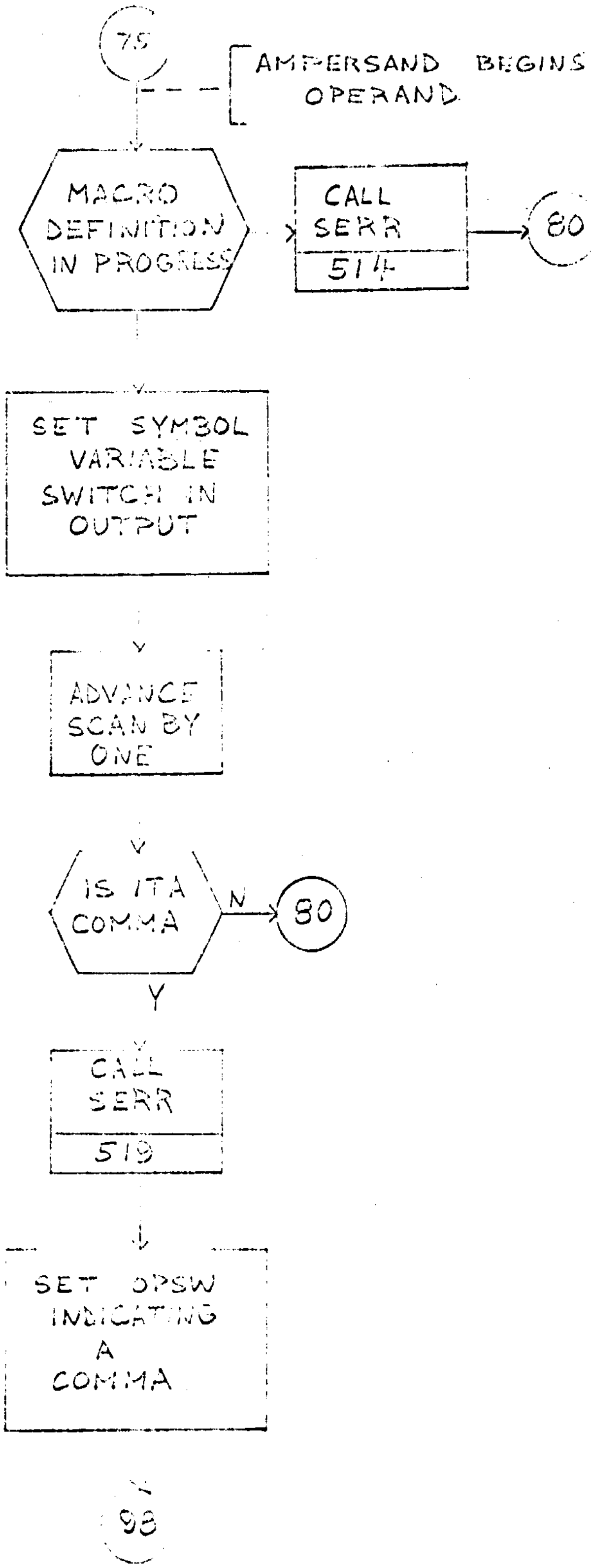

60
SCAN FOR
OP-CODE
ADVANCE
SCAN
BY ONE
HAS SCAN
GONE BEYOND
COLUMN 72
Y
CALL
SERR
510
85
N
IS IT A
BLANK
Y
60
N
IS IT AN
AMPERSAND
Y
68
N
61

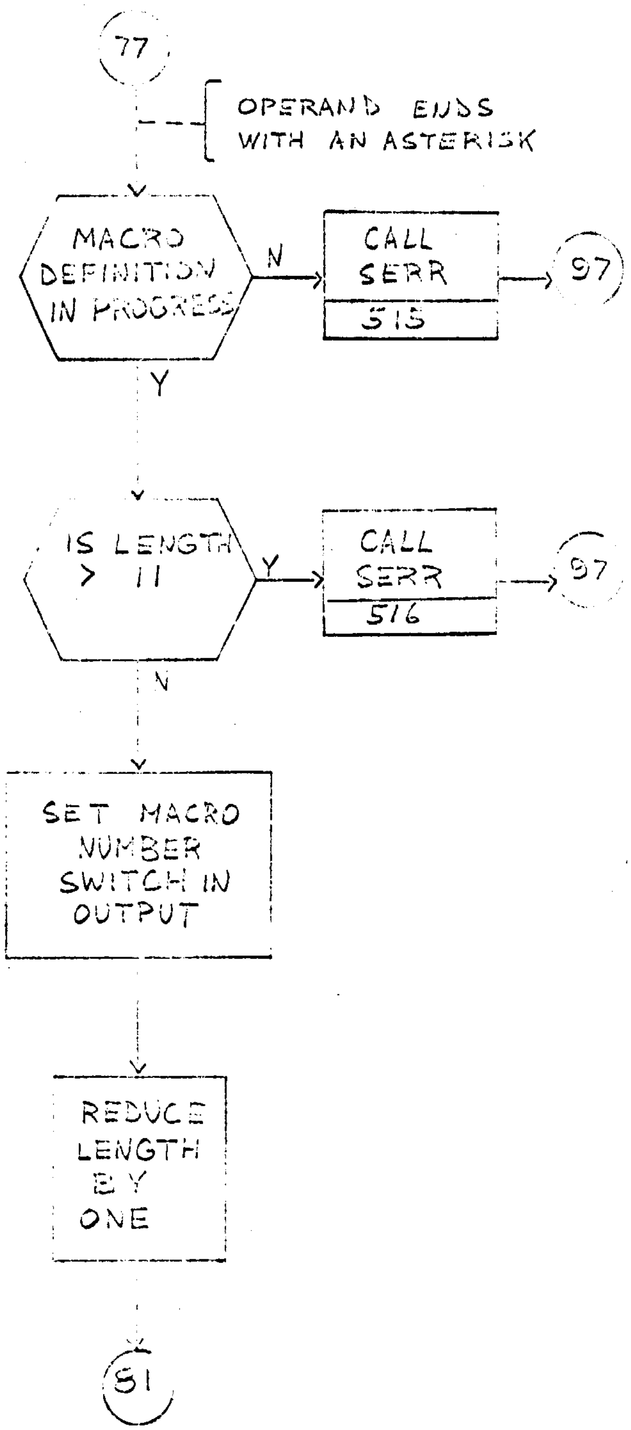

61
PICK UP OP-CODE
STORE START OF OP-CODE IN NSTART
62
ADVANCE SCAN BY ONE
HAS SCAN GONE BEYOND COLUMN 72
Y
CALL SERR
510
85
N
IS IT A COMMA
Y
CALL SERR
517
95
N
IS IT A BLANK
N
62
Y

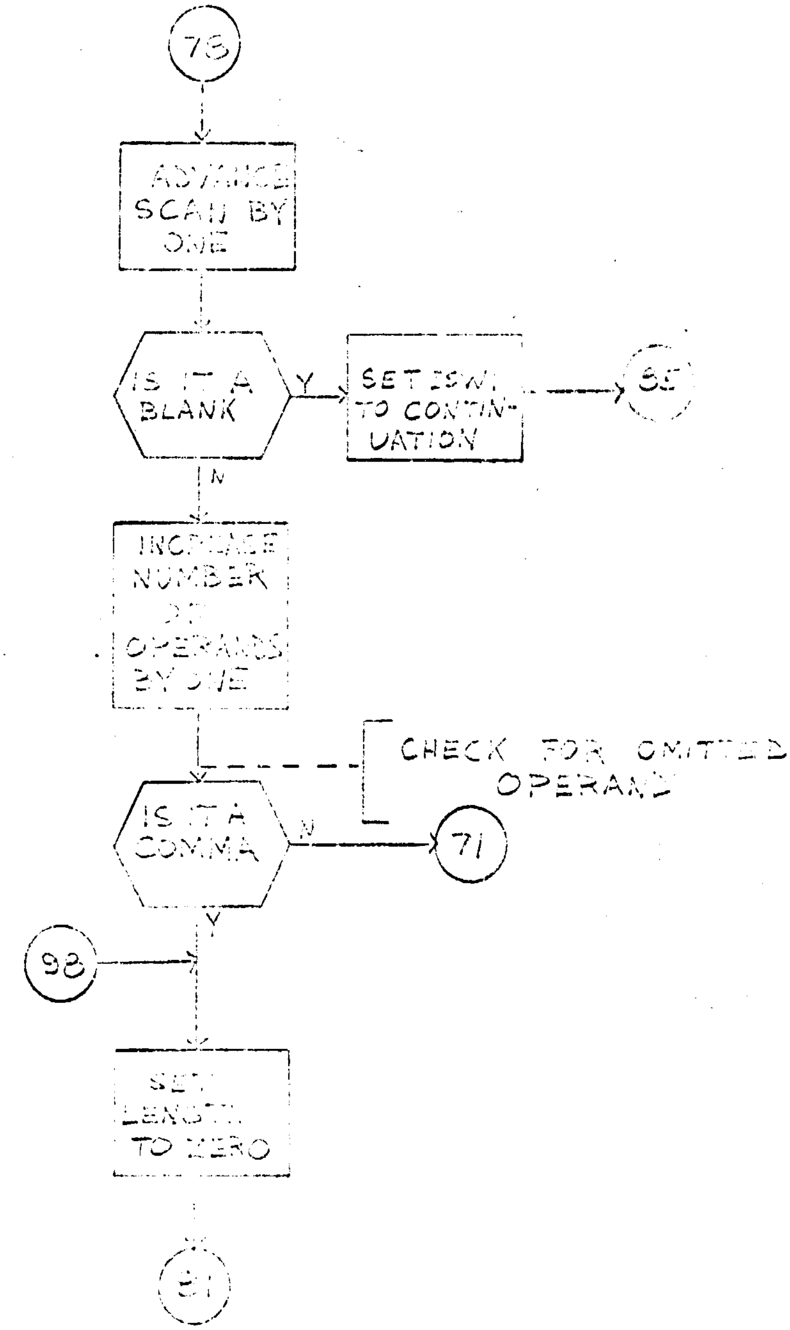

61A
COMPUTE LENGTH OF OP-CODE
IS LENGTH > 12
Y
CALL SERR
519
93
N
STORE OP-CODE AND LENGTH IN OUTPUT
IS OPERAND WITHIN 10 COLUMNS OF OP-CODE
N
RETURN
Y
70

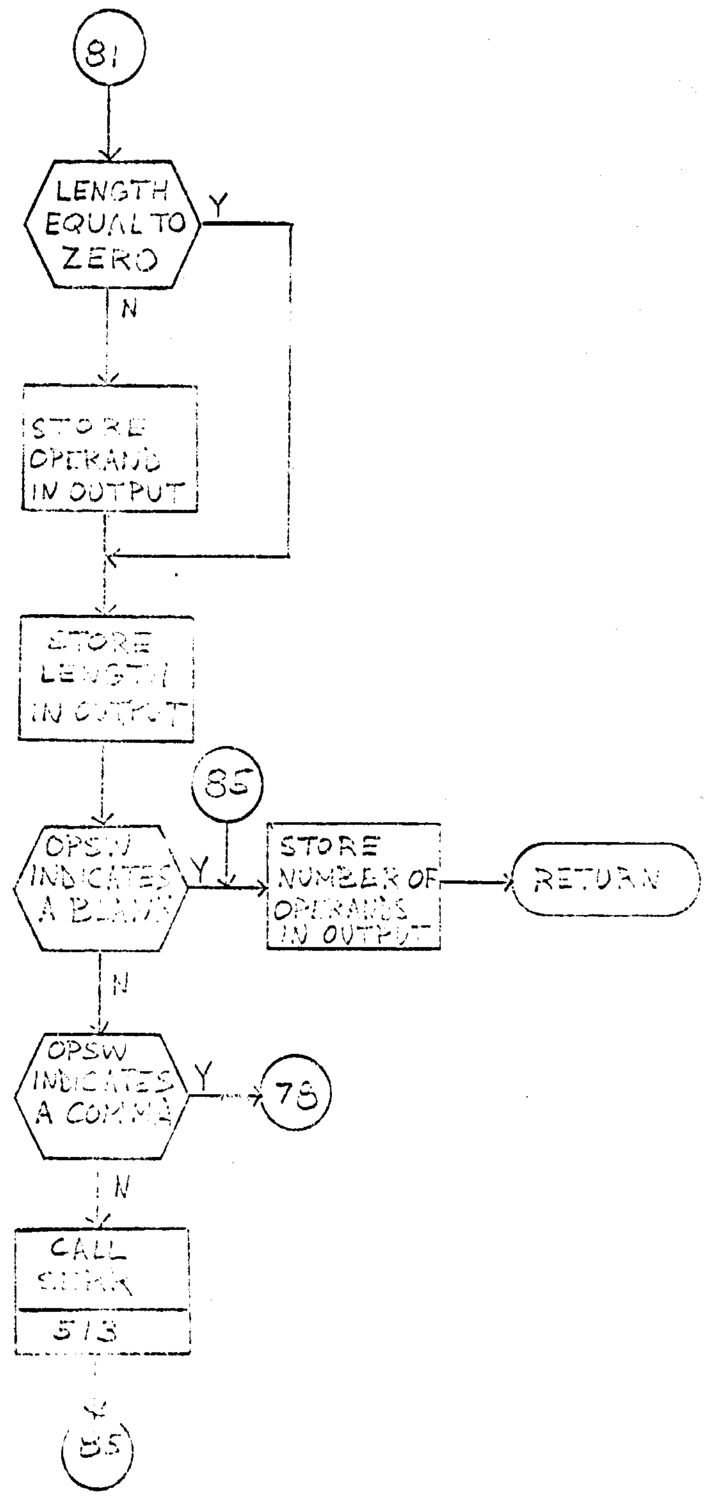

68
AMPERSAND BEGINS OP-CODE
MACRO DEFINITION IN PROGRESS
N
Y
CALL SERR
511
61
SET OP-CODE VARIABLE SWITCH IN OUTPUT
ADVANCE SCAN BY ONE
61

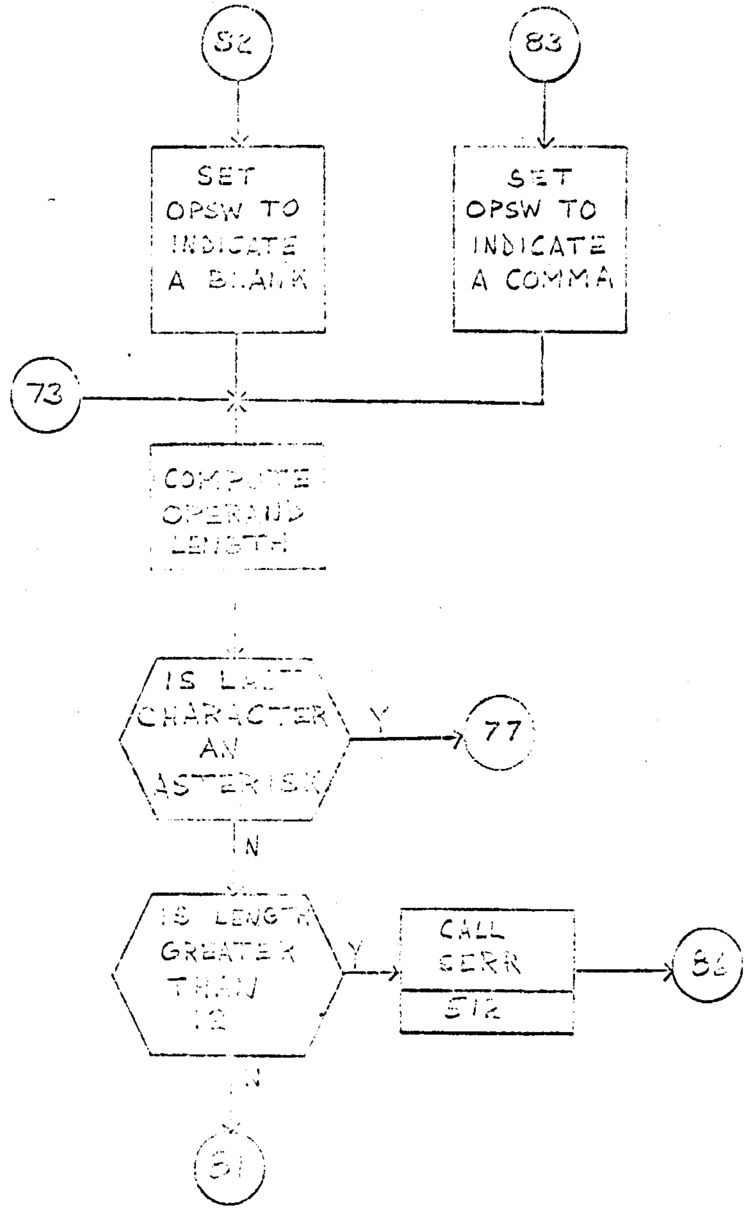

70
OPERAND PICKUP
SET OPERAND COUNT TO ZERO
71
IS FIRST CHARACTER OF OPERAND AN AMPERSAND
Y
75
N
80
STORE BEGINNING OF OPERAND IN NSTART
72
ADVANCE SCAN BY ONE
IS IT A BLANK
Y
82
N
IS IT A COMMA
Y
82
N
72

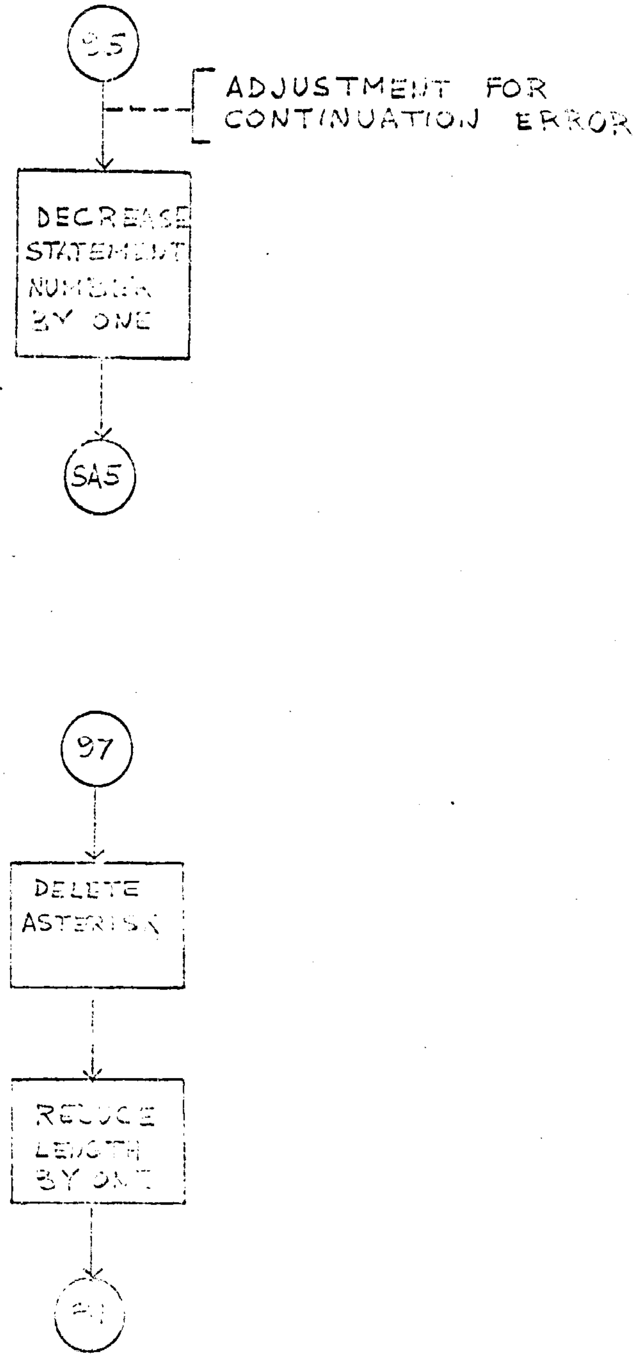

75
AMPERSAND BEGINS OPERAND
MACRO DEFINITION IN PROGRESS
CALL SERR
514
80
SET SYMBOL VARIABLE SWITCH IN OUTPUT
ADVANCE SCAN BY ONE
IS IT A COMMA
N
80
Y
CALL SERR
519
SET OPSW INDICATING A COMMA
98

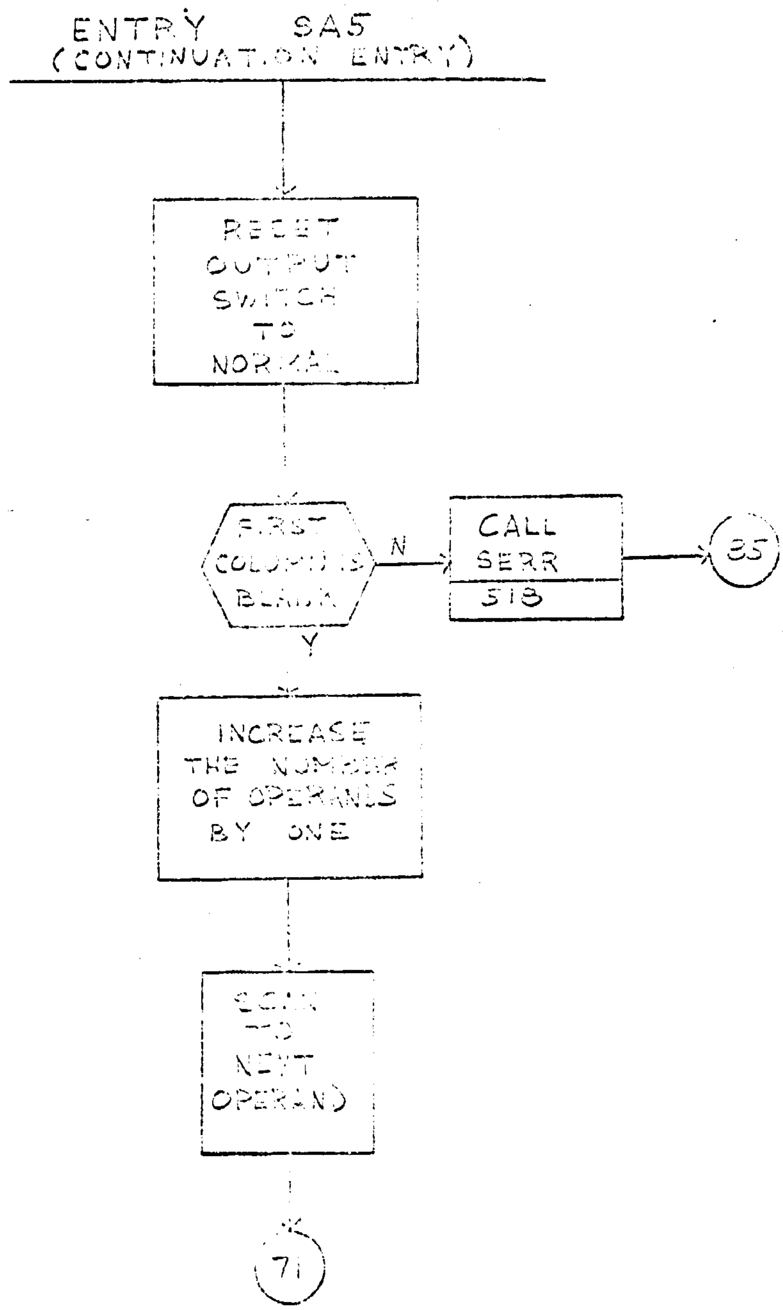

77
OPERAND ENDS WITH AN ASTERISK
MACRO DEFINITION IN PROGRESS
N
CALL SERR
515
97
Y
IS LENGTH > 11
Y
CALL SERR
516
97
N
SET MACRO NUMBER SWITCH IN OUTPUT
REDUCE LENGTH BY ONE
81


78
ADVANCE SCAN BY ONE
IS IT A BLANK
Y
SET ISW TO CONTIN-UATION
85
N
INCREASE NUMBER OF OPERANDS BY ONE
CHECK FOR OMITTED OPERAND
IS IT A COMMA
N
71
Y
98
SET LENGTH TO ZERO
81


81
LENGTH EQUAL TO ZERO
Y
N
STORE OPERAND IN OUTPUT
STORE LENGTH IN OUTPUT
85
OPSW INDICATES A BLANK
Y
STORE NUMBER OF OPERANDS IN OUTPUT
RETURN
N
OPSW INDICATES A COMMA
Y
78
N
CALL
513
85


82
83
SET OPSW TO INDICATE A BLANK
SET OPSW TO INDICATE A COMMA
73
COMPUTE OPERAND LENGTH
IS LAST CHARACTER AN ASTERISK
Y
77
N
IS LENGTH GREATER THAN 12
Y
CALL SERR
512
86
N
51


95
ADJUSTMENT FOR CONTINUATION ERROR
DECREASE STATEMENT NUMBER BY ONE
SA5
97
DELETE ASTERISK
REDUCE LENGTH BY ONE

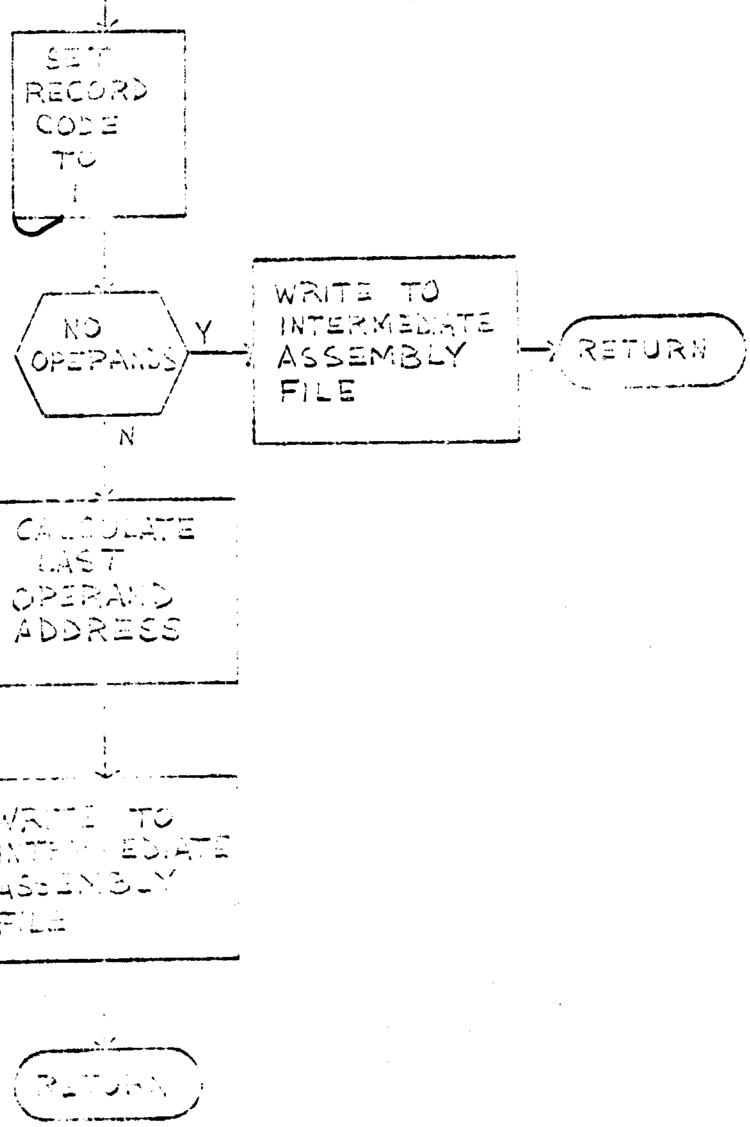


ENTRY SA5
(CONTINUATION ENTRY)
RESET OUTPUT SWITCH TO NORMAL
N
CALL SERR
518
85
Y
INCREASE THE NUMBER OF OPERANDS BY ONE
SCAN TO NEXT OPERAND
71

SUBROUTINE SA6

This routinc is used to write records to the intermediate assembly file. It is used by the ASM1 routine. The first word of each record contains a count of the number of words in the record. The records are therefore of variable length.

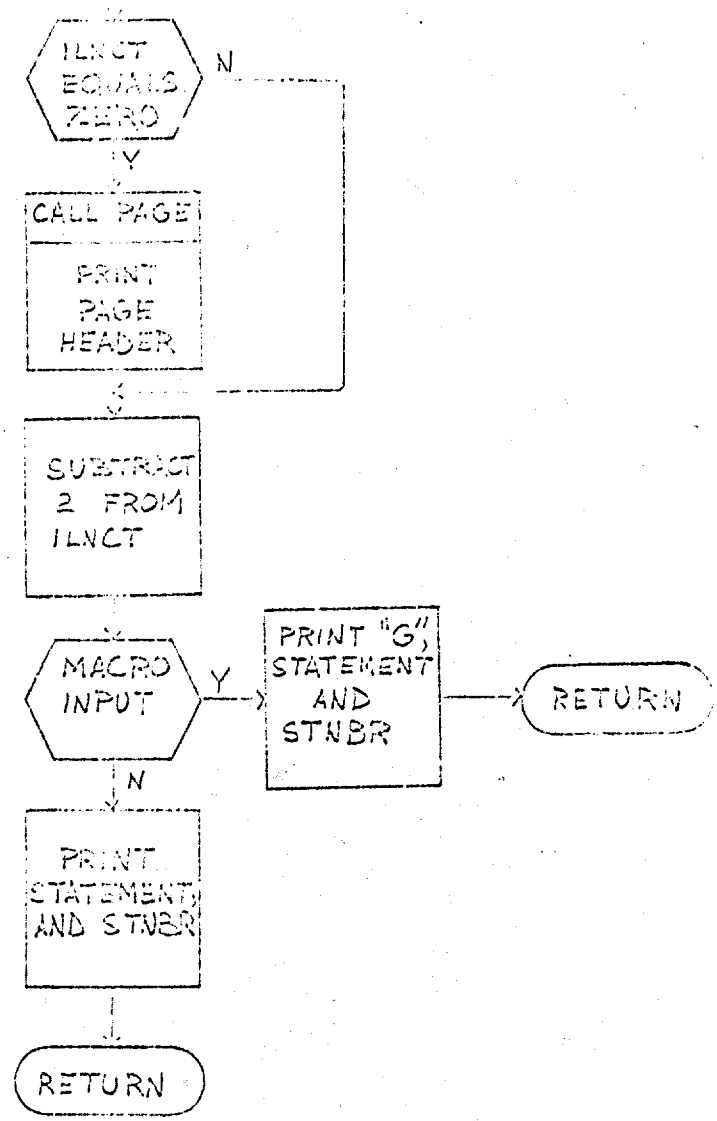
SUBROUTINE SA6
(INTERMEDIATE OUTPUT)



SUBROUTINE SA7

This routine is used by the ASM1 routine to print the input statements. The routine provides two auxiliary functions. The first of these is that a check is made of the number of lines written on the page. If the number of lines has been used up, the PAGE routine is called. The second auxiliary function consists of a check to see if the input is coming from the macro input file, corresponding to the expansion of a macro call, in which case a G precedes the print out of the statement.

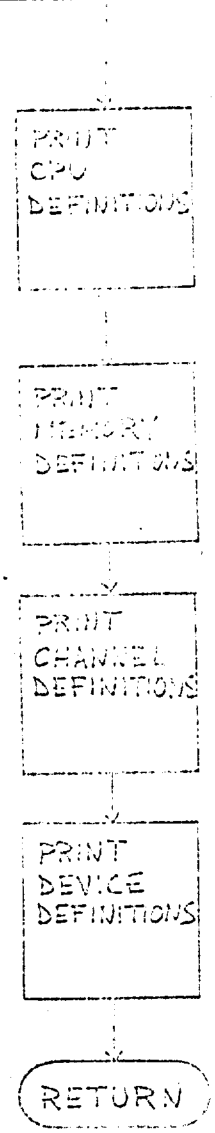
SUBROUTINE SA7
(PRINT INPUT)



SUBROUTINE SA8

This routine prints out the hardware definitions at the end of the first phase of the first pass subroutine. The CPU, MEMORY, CHANNEL, and DEVICE definitions are printed out on the system printer.

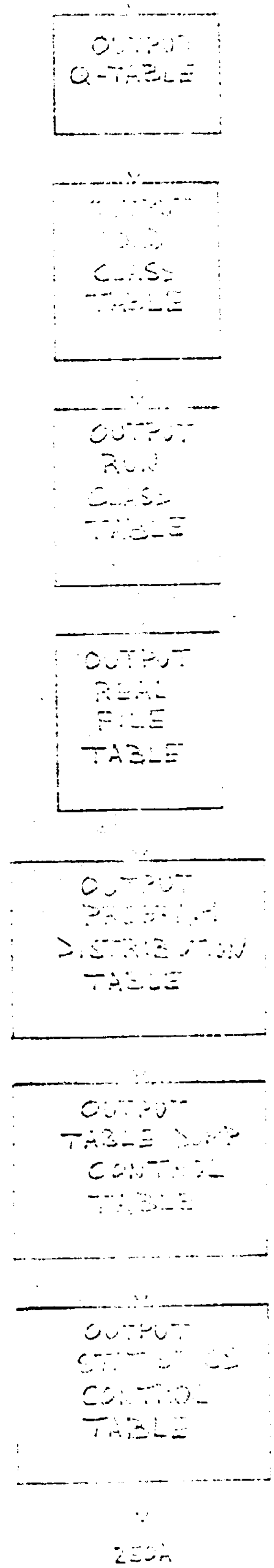
SUBROUTINE SAB
(PRINT HARDWARE TABLES)

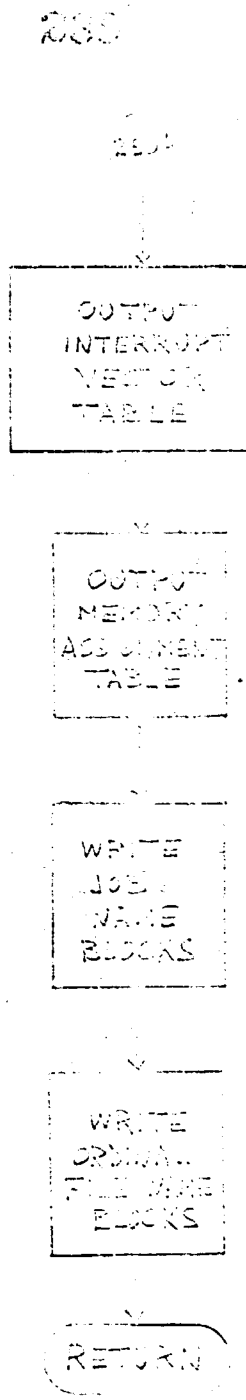


SUBROUTINE SA9

This routine is called by the first pass subroutine to output hardware and program definition tables to the simulator input tape. The queue tables, load class tables, run class tables. real file tables, program distribution table, table dump control table, output statistics control, output interrupt vector table, output memory assignment table, are written to the simulator input tape. In addition, the job name and ordinal file name blocks are written to the statistics tape.

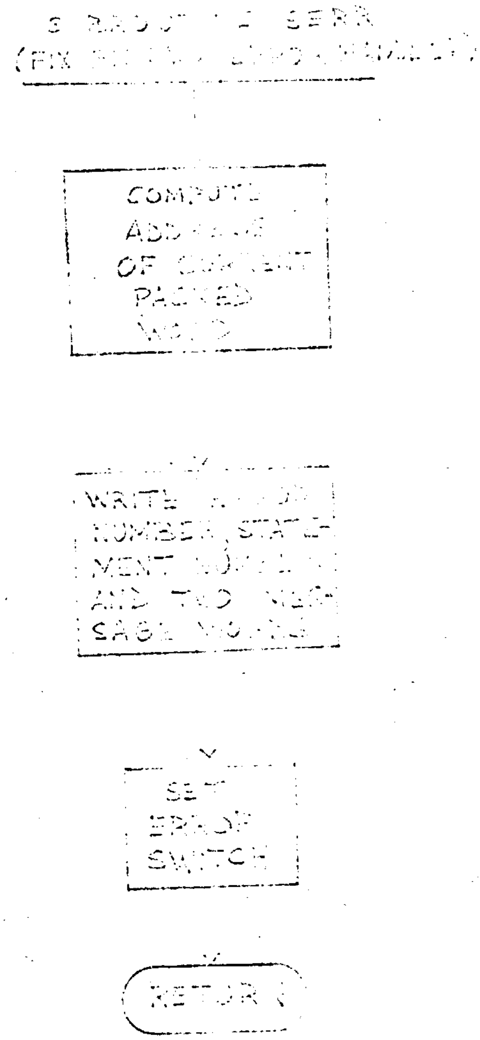
SUBROUTINE SAB
(OUTPUT TABLES)




OUTPUT INTERRUPT VECTOR TABLE
OUTPUT MEMORY ASSIGNMENT TABLE
WRITE JOB NAME BLOCKS
WRITE ORDINAL FILE NAME BLOCKS
RETURN

SUBROUTINE SERR

This is the error routine for the fix fields routine. It does the same functions as the NERR routine, only it automatically computes the current location of the fix fields scan and includes the current two words to be written onto the error file. These words will then be used in the error message as it is printed out by the error pass. The master error switch is turned ON.

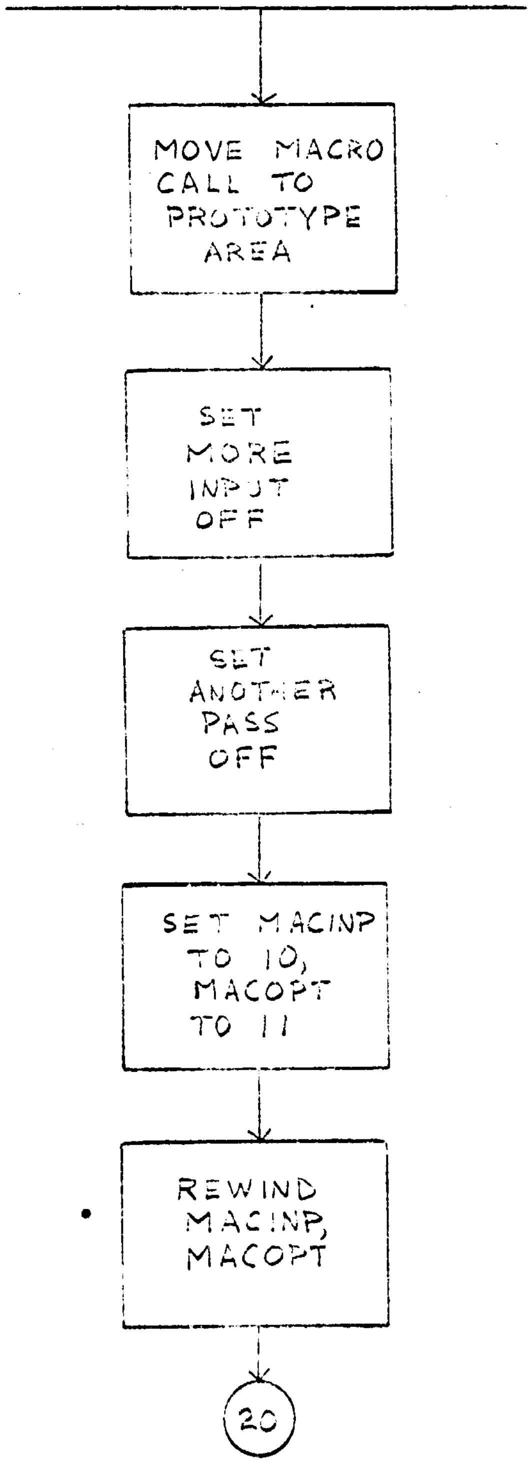

SERR
COMPUTE
OF
PACKED
WRITE
NUMBER, STATE-
MENT
AND TWO
SET
ERROR
SWITCH
RETURN

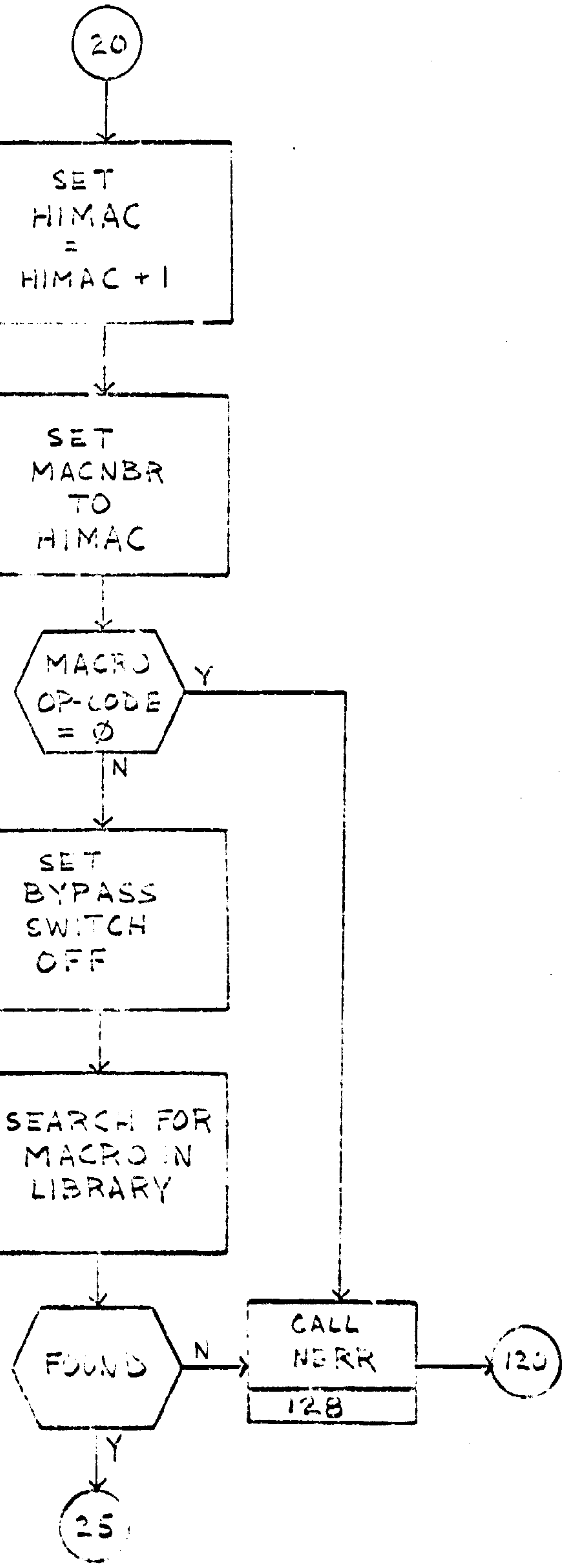
SUBROUTINE SMACRO

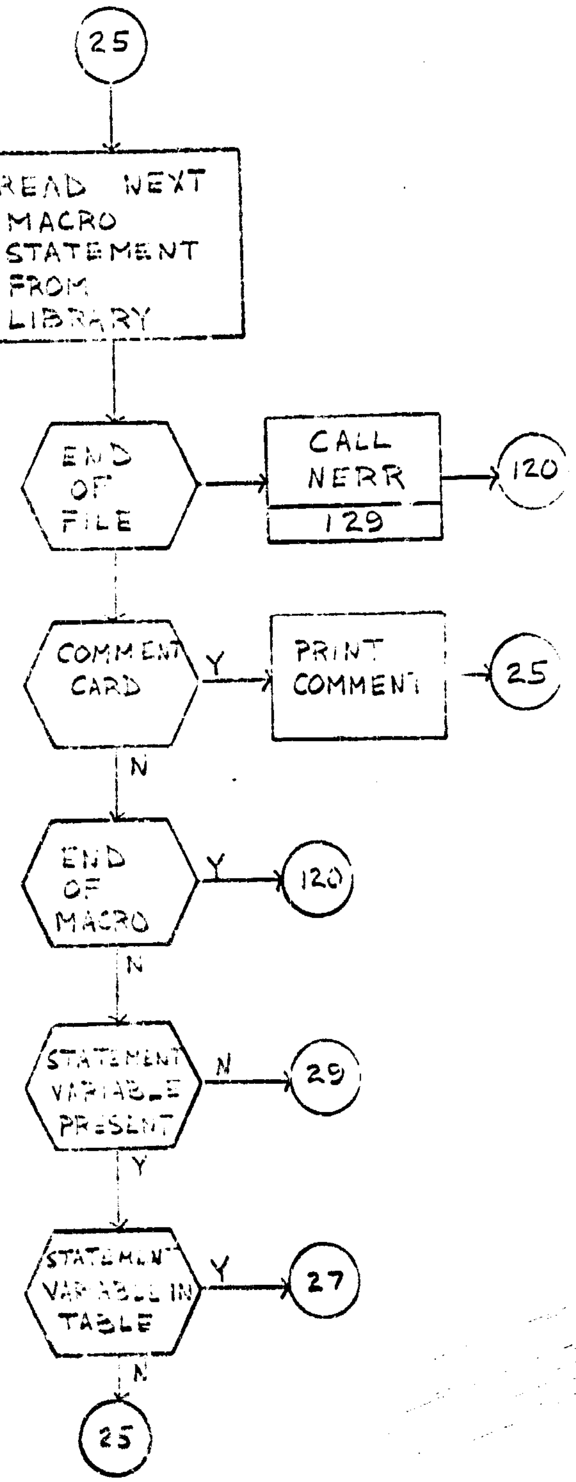
This routine is called to process a macro call appearing in job source input. This routine expands the macro call, processing nested macro calls as they occur, and places the generated statements into a macro input file. A switch is then set which signals the first pass subroutine that is now to take its input from the macro input file.

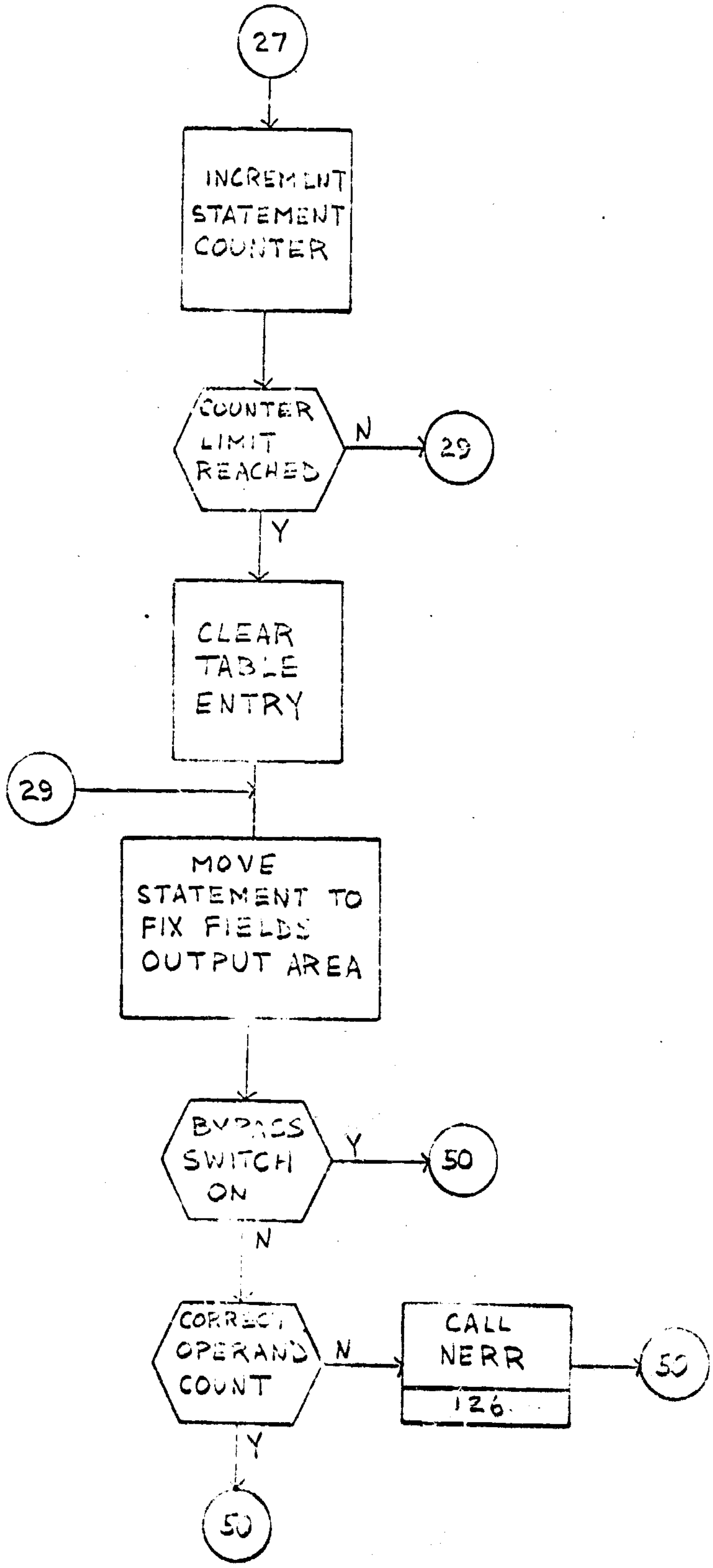
The SMACRO subroutine first moves the macro call to a prototype area. It then performs some house-keeping, initializing itself and setting values for macro numbers which must be appended to statement labels. The library is then searched for the macro call name, and the macro is read from the library and stored on the macro input file. Statement label variables, and statement labels requiring macro numbers to be appended are processed as required.

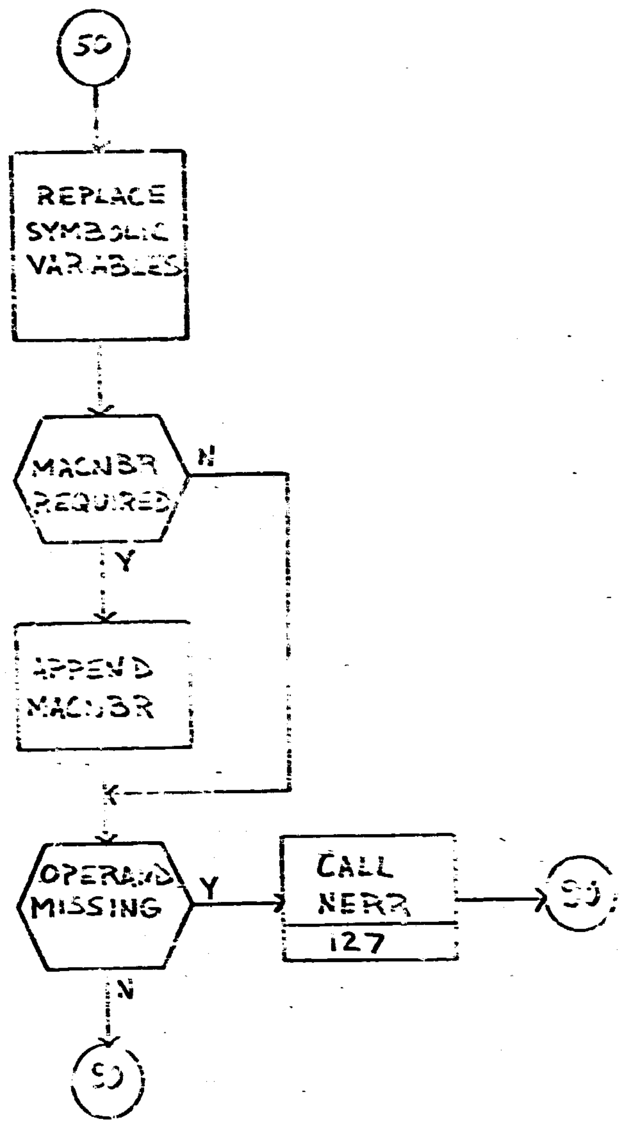
SUBROUTINE SMACRO
(PROCESS MACRO)

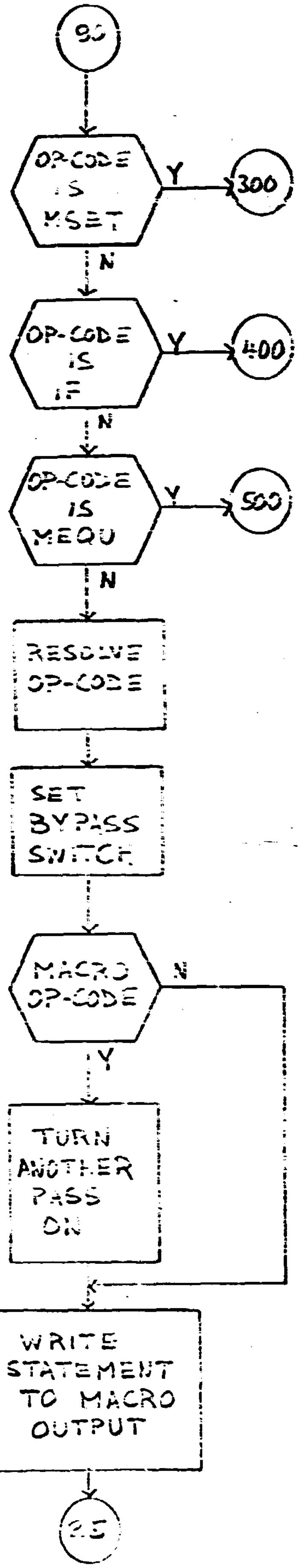


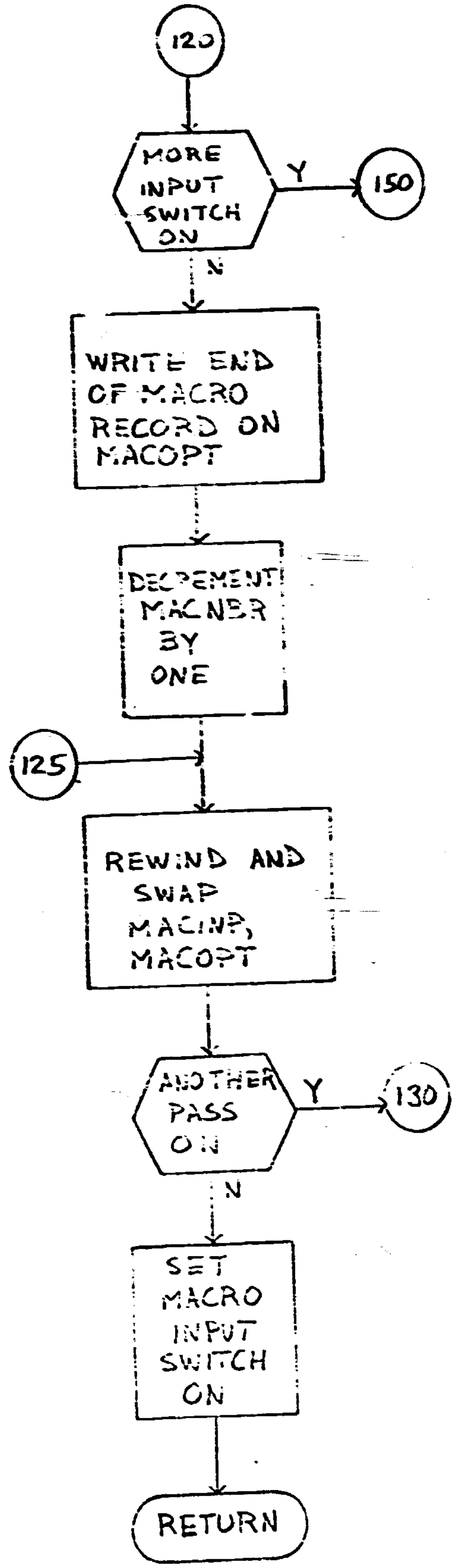

20
SET HIMAC = HIMAC + 1
SET MACNBR TO HIMAC
MACRO OP-CODE = Ø
Y
N
SET BYPASS SWITCH OFF
SEARCH FOR MACRO IN LIBRARY
FOUND
N
Y
CALL NERR
128
120
25

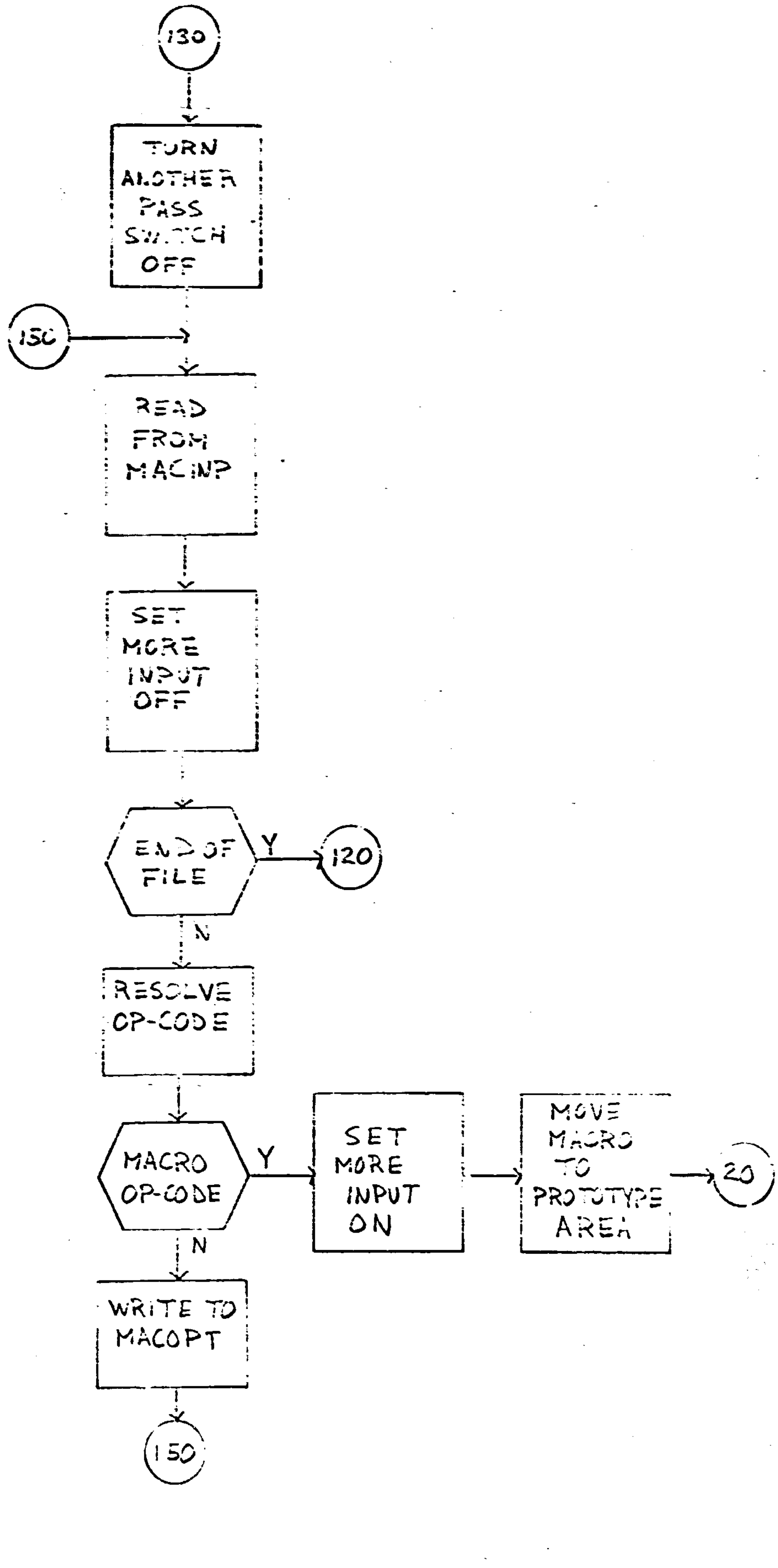

25
READ NEXT MACRO STATEMENT FROM LIBRARY
END OF FILE
CALL NERR
129
120
COMMENT CARD
Y
PRINT COMMENT
25
N
END OF MACRO
Y
120
N
STATEMENT VARIABLE PRESENT
N
29
Y
STATEMENT VARIABLE IN TABLE
Y
27
N
25

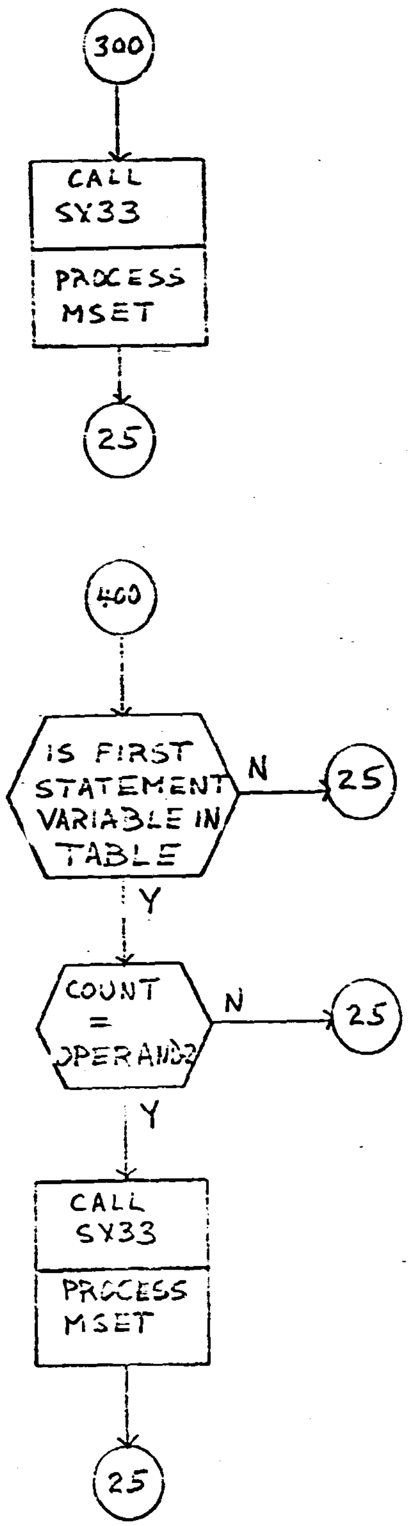

27
INCREMENT
STATEMENT
COUNTER
COUNTER
LIMIT
REACHED
N
29
Y
CLEAR
TABLE
ENTRY
29
MOVE
STATEMENT TO
FIX FIELDS
OUTPUT AREA
BYPASS
SWITCH
ON
Y
50
N
CORRECT
OPERAND
COUNT
N
CALL
NERR
126
50
Y
50

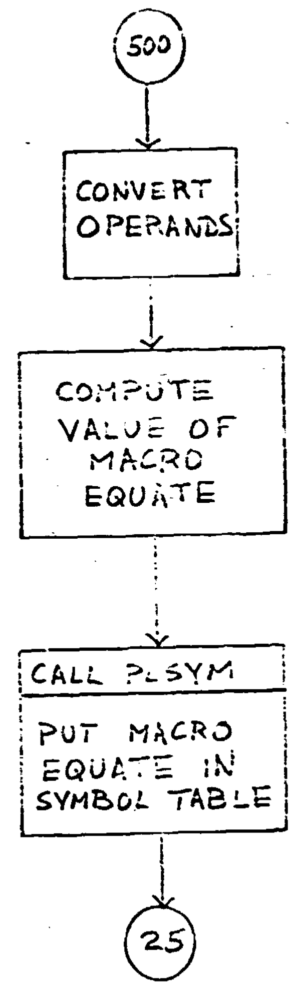

50
REPLACE SYMBOLIC VARABLES
MACNBR REQUIRED
N
Y
APPEND MACNBR
OPERAND MISSING
Y
CALL NERR
127
90
N
90


OP-CODE IS MSET
Y
300
N
OP-CODE IS IF
Y
400
N
OP-CODE IS MEQU
Y
500
N
RESOLVE OP-CODE
SET BYPASS SWITCH
MACRO OP-CODE
N
Y
TURN ANOTHER PASS ON
WRITE STATEMENT TO MACRO OUTPUT


120
MORE INPUT SWITCH ON
Y
150
N
WRITE END OF MACRO RECORD ON MACOPT
DECREMENT MACNBR BY ONE
125
REWIND AND SWAP MACINP, MACOPT
ANOTHER PASS ON
Y
130
N
SET MACRO INPUT SWITCH ON
RETURN


130
TURN ANOTHER PASS SWITCH OFF
150
READ FROM MACINP
SET MORE INPUT OFF
END OF FILE
Y
120
N
RESOLVE OP-CODE
MACRO OP-CODE
Y
SET MORE INPUT ON
MOVE MACRO TO PROTOTYPE AREA
20
N
WRITE TO MACOPT
150


300
CALL SY33
PROCESS MSET
25
400
IS FIRST STATEMENT VARIABLE IN TABLE
N
25
Y
COUNT = OPERAND2
N
25
Y
CALL SY33
PROCESS MSET
25

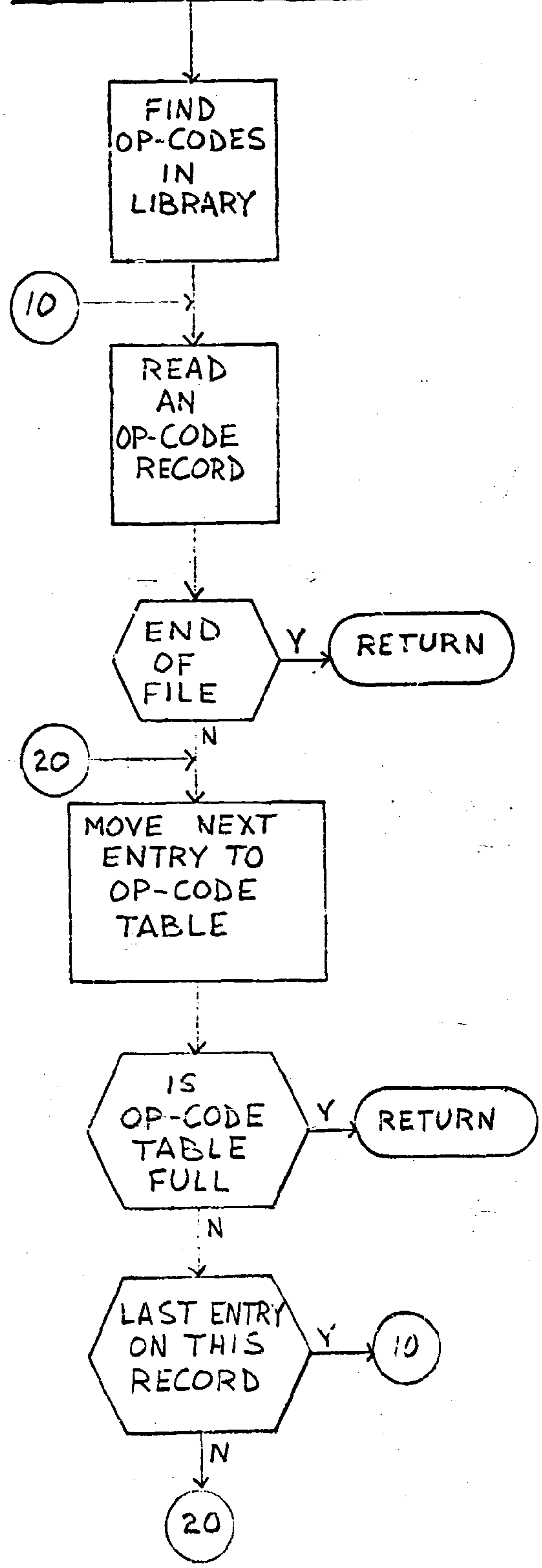


500
CONVERT OPERANDS
COMPUTE VALUE OF MACRO EQUATE
CALL PLSYM
PUT MACRO EQUATE IN SYMBOL TABLE
25

SUBROUTINE SOPCD

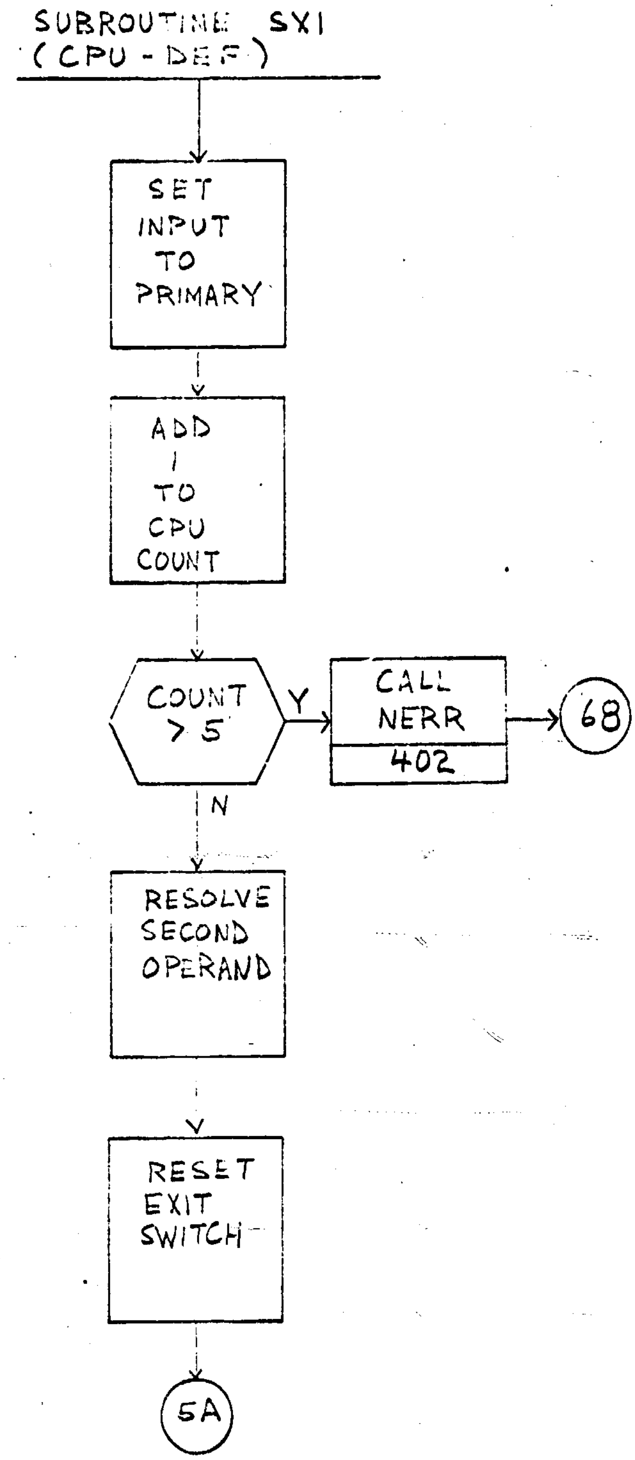
This routine searches the library for the op-codes. After finding them, it reads them into the op-code table in memory where it is used for processing by the first pass. After reading the formatted and packed op-codes and loading them unpacked into the table, the remainder of the table is filled with all nines to insure a correct binary search of the table for op-codes.

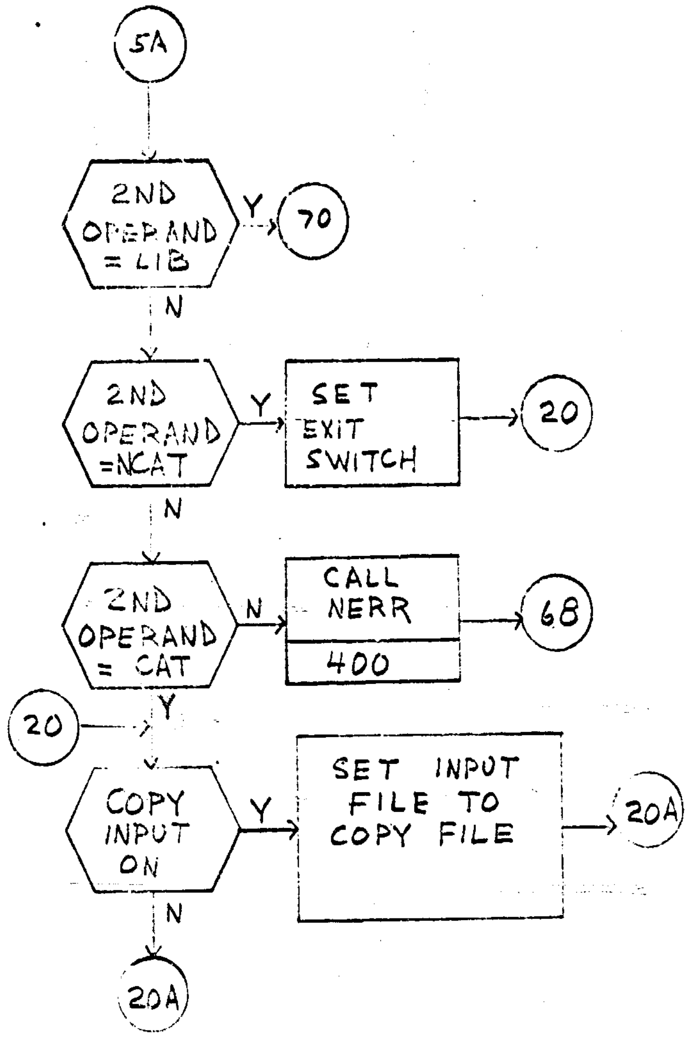
SUBRØUTINE: SØPCD
(LØAD ØP-CØDE TABLE)

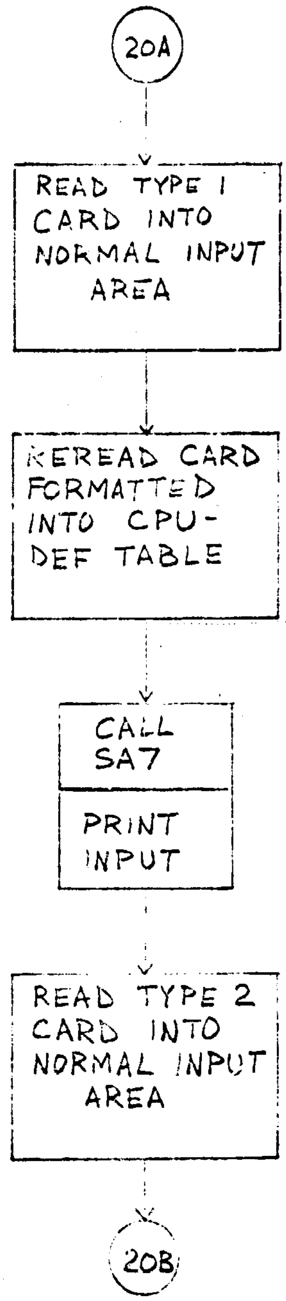


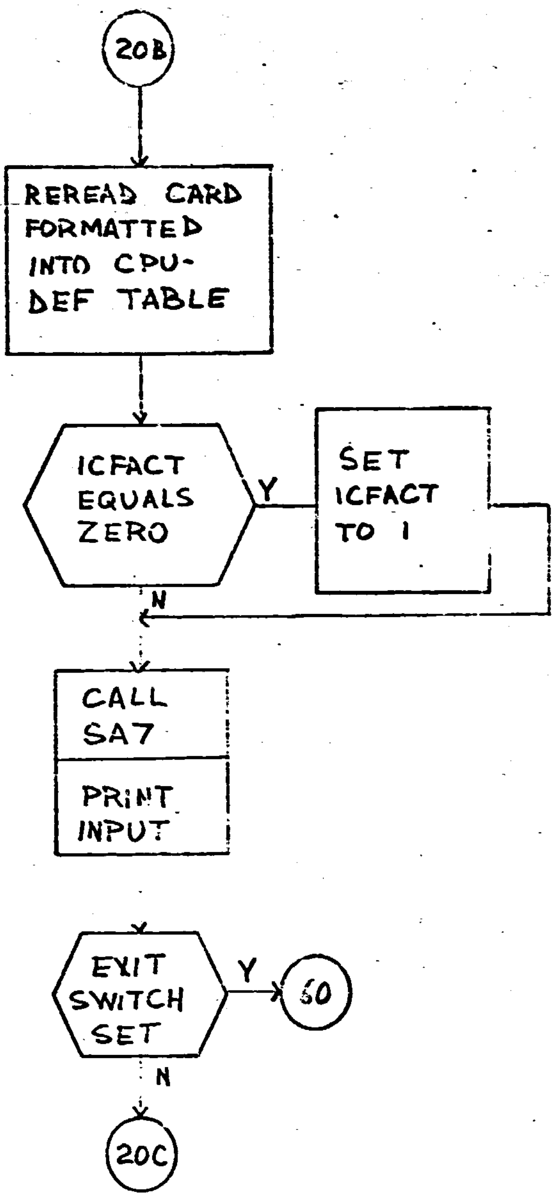
SUBROUTINE SX1

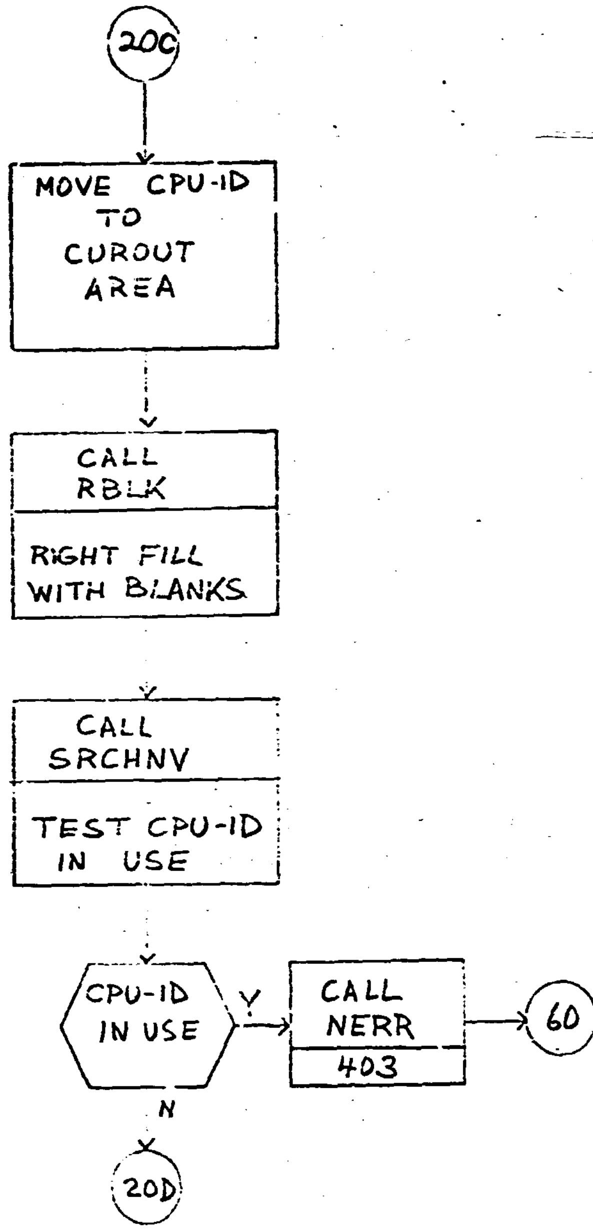
This routine processes CPU definitions. Within this routine, a test is made of the second operand, which indicates whether the CPU definition is to be found in the library, or if it is in the input stream, whether it is to be catalogued or not. The CPU definition is then read from the library or from the input stream as required, and placed into the appropriate table in common. The CPU definition is printed out as it is received. A count is maintained of the number of such CPU definitions received. A check is made to determine if the maximum number has been exceeded.

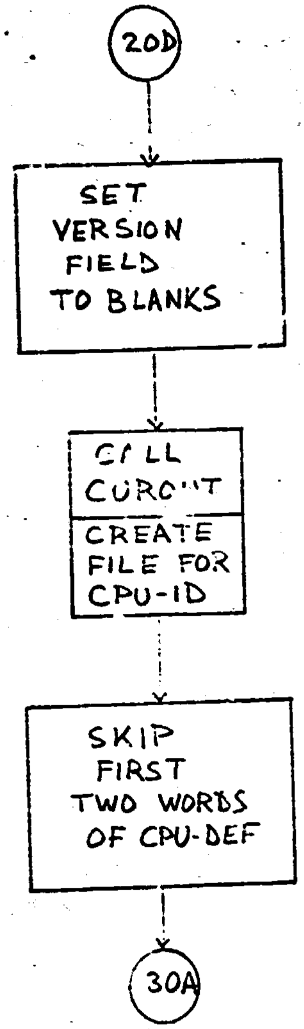

SUBROUTINE SXI
(CPU-DEF)
SET INPUT TO PRIMARY
ADD 1 TO CPU COUNT
COUNT > 5
Y
CALL NERR
402
68
N
RESOLVE SECOND OPERAND
RESET EXIT SWITCH
5A

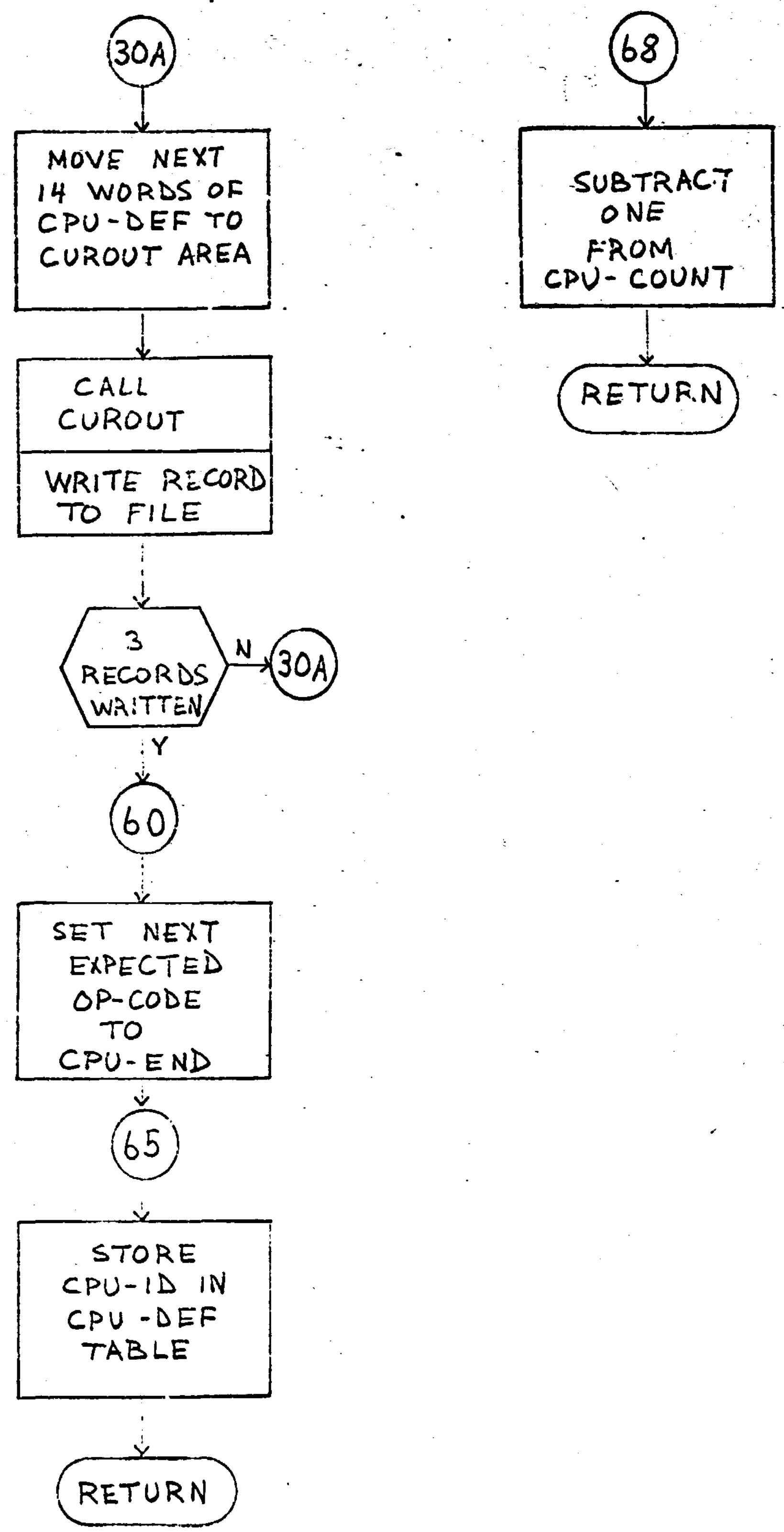

5A
2ND OPERAND = LIB
Y
70
N
2ND OPERAND =NCAT
Y
SET EXIT SWITCH
20
N
2ND OPERAND = CAT
N
CALL NERR
400
68
Y
20
COPY INPUT ON
Y
SET INPUT FILE TO COPY FILE
20A
N
20A

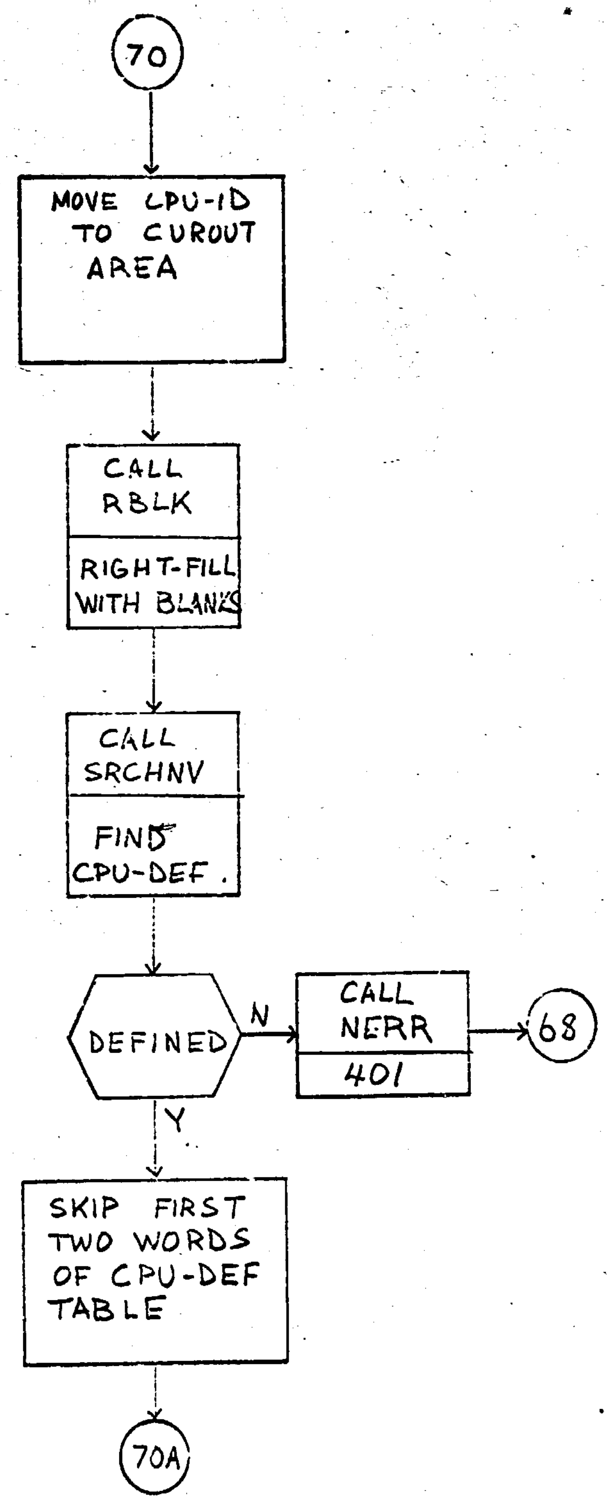

20A
READ TYPE 1 CARD INTO NORMAL INPUT AREA
REREAD CARD FORMATTED INTO CPU-DEF TABLE
CALL SA7
PRINT INPUT
READ TYPE 2 CARD INTO NORMAL INPUT AREA
20B

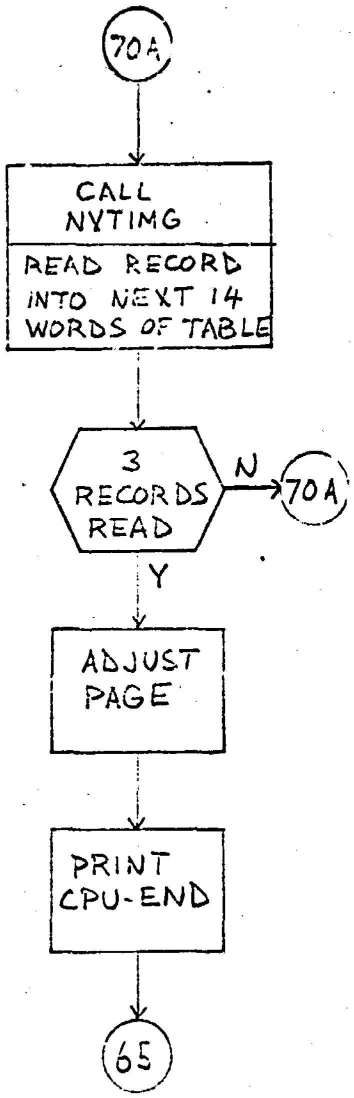

20B
REREAD CARD FORMATTED INTO CPU-DEF TABLE
ICFACT EQUALS ZERO
Y
SET ICFACT TO 1
N
CALL SA7
PRINT INPUT
EXIT SWITCH SET
Y
60
N
20C


20C
MOVE CPU-ID TO CUROUT AREA
CALL RBLK
RIGHT FILL WITH BLANKS
CALL SRCHNV
TEST CPU-ID IN USE
CPU-ID IN USE
Y
CALL NERR
403
60
N
20D


20D
SET VERSION FIELD TO BLANKS
CALL
CREATE FILE FOR CPU-ID
SKIP FIRST TWO WORDS OF CPU-DEF
30A


30A
MOVE NEXT 14 WORDS OF CPU-DEF TO CUROUT AREA
CALL CUROUT
WRITE RECORD TO FILE
3 RECORDS WRITTEN
N
30A
Y
60
SET NEXT EXPECTED OP-CODE TO CPU-END
65
STORE CPU-ID IN CPU-DEF TABLE
RETURN
68
SUBTRACT ONE FROM CPU-COUNT
RETURN

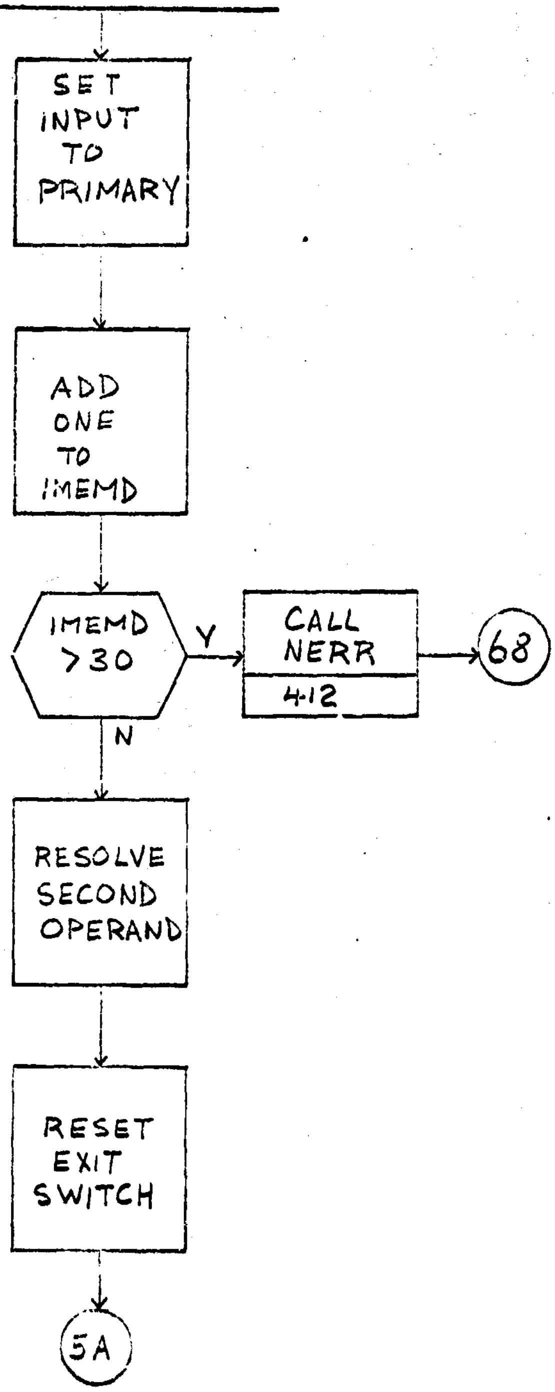

70
MOVE CPU-ID TO CUROUT AREA
CALL RBLK
RIGHT-FILL WITH BLANKS
CALL SRCHNV
FIND CPU-DEF.
DEFINED
N
Y
CALL NERR
401
68
SKIP FIRST TWO WORDS OF CPU-DEF TABLE
70A

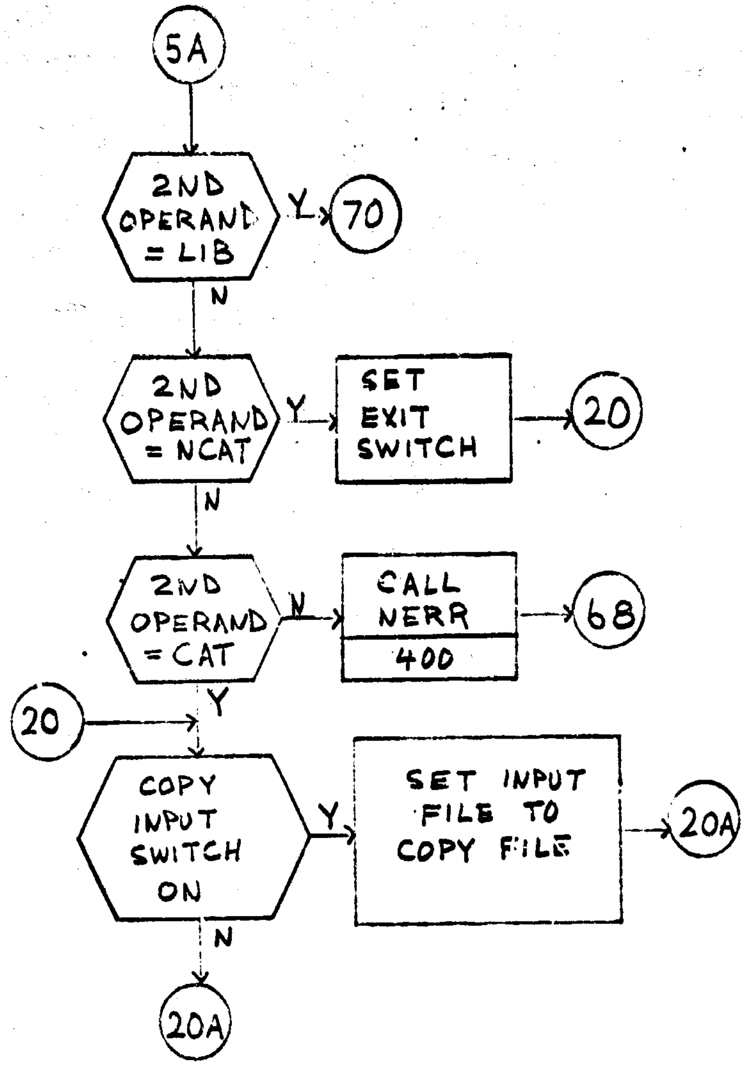

70A
CALL NYTIMG
READ RECORD INTO NEXT 14 WORDS OF TABLE
3 RECORDS READ
N
70A
Y
ADJUST PAGE
PRINT CPU-END
65

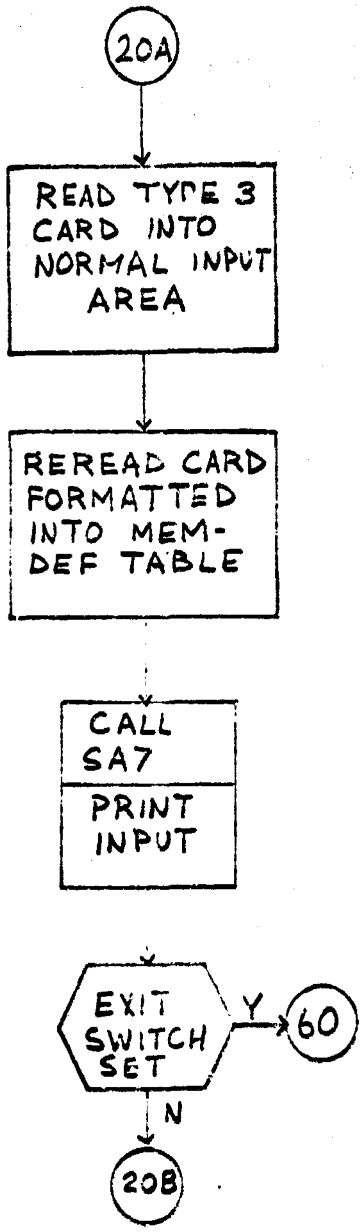
SUBROUTINE SX2

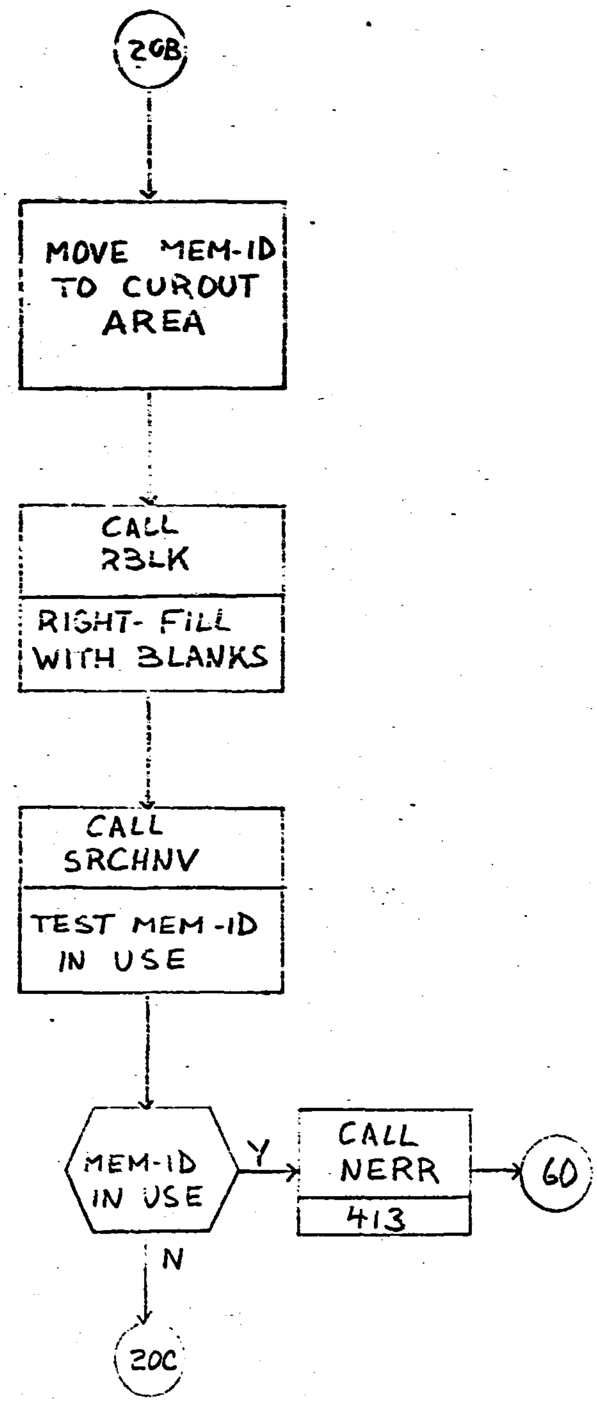
This routine processes memory definitions. The second operand is checked to see whether the definition is in the library or, if it is in the input stream, whether it is to be catalogued or not. The memory definition is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. A count is made of the number of memory definitions. A check is made to see if the number of memory definitions has exceeded the maximum permitted.

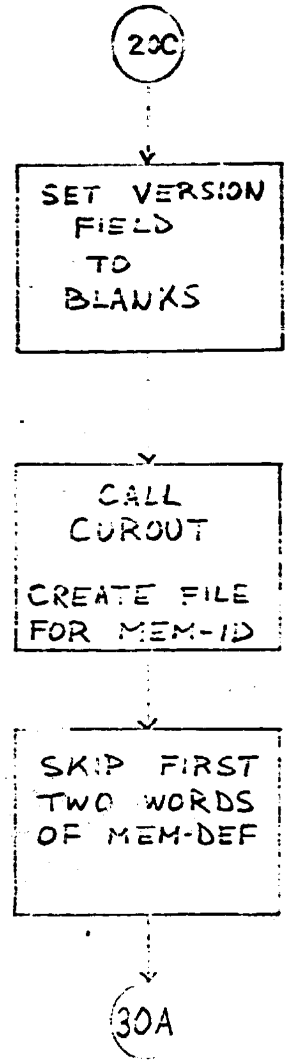
SUBROUTINE SX2
(MEM-DEF)

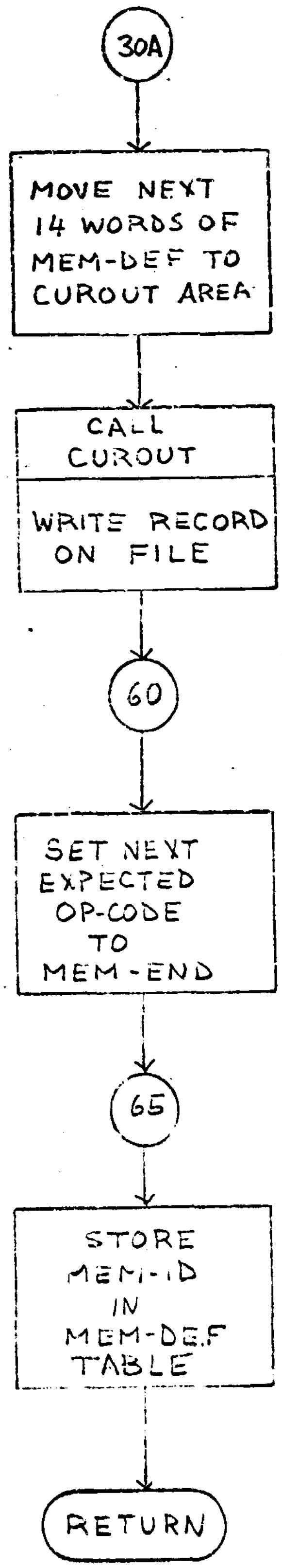


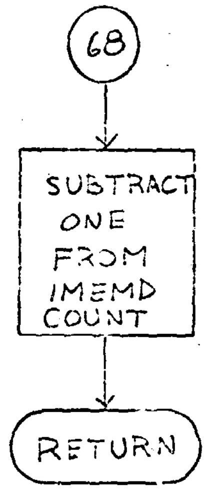

5A
2ND OPERAND = LIB
Y
70
N
2ND OPERAND = NCAT
Y
SET EXIT SWITCH
20
N
2ND OPERAND = CAT
N
CALL NERR
400
68
Y
20
COPY INPUT SWITCH ON
Y
SET INPUT FILE TO COPY FILE
20A
N
20A

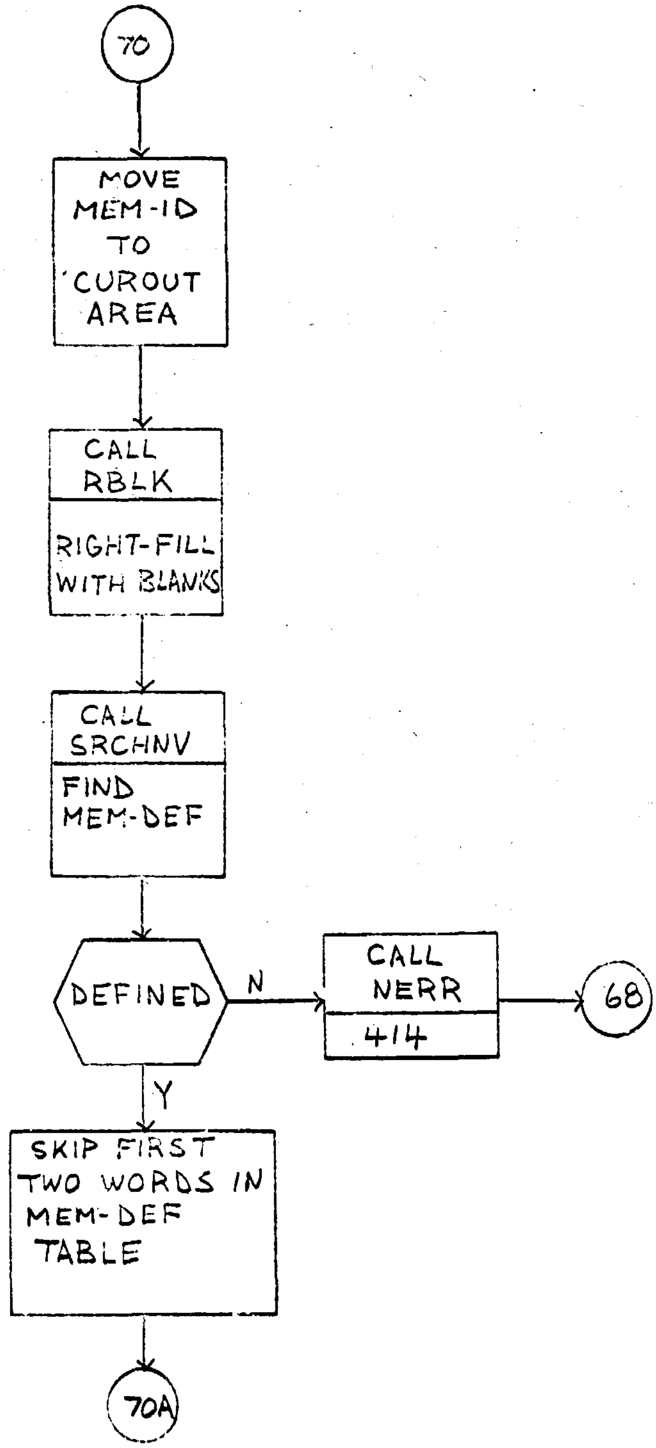

20A
READ TYPE 3 CARD INTO NORMAL INPUT AREA
REREAD CARD FORMATTED INTO MEM-DEF TABLE
CALL SA7
PRINT INPUT
EXIT SWITCH SET
Y
60
N
20B

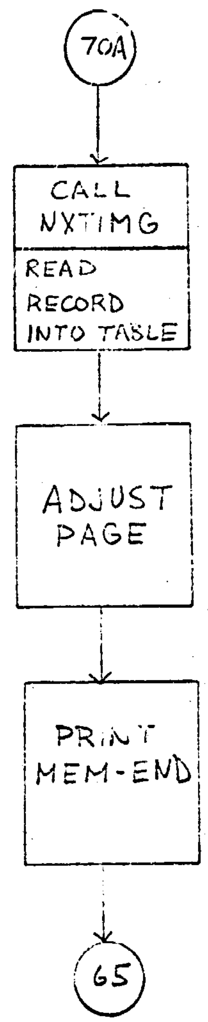

20B
MOVE MEM-ID TO CUROUT AREA
CALL RBLK
RIGHT-FILL WITH BLANKS
CALL SRCHNV
TEST MEM-ID IN USE
MEM-ID IN USE
Y
CALL NERR
413
60
N
20C


20C
SET VERSION FIELD TO BLANKS
CALL CUROUT
CREATE FILE FOR MEM-ID
SKIP FIRST TWO WORDS OF MEM-DEF
30A


30A
MOVE NEXT 14 WORDS OF MEM-DEF TO CUROUT AREA
CALL CUROUT
WRITE RECORD ON FILE
60
SET NEXT EXPECTED OP-CODE TO MEM-END
65
STORE MEM-ID IN MEM-DEF TABLE
RETURN

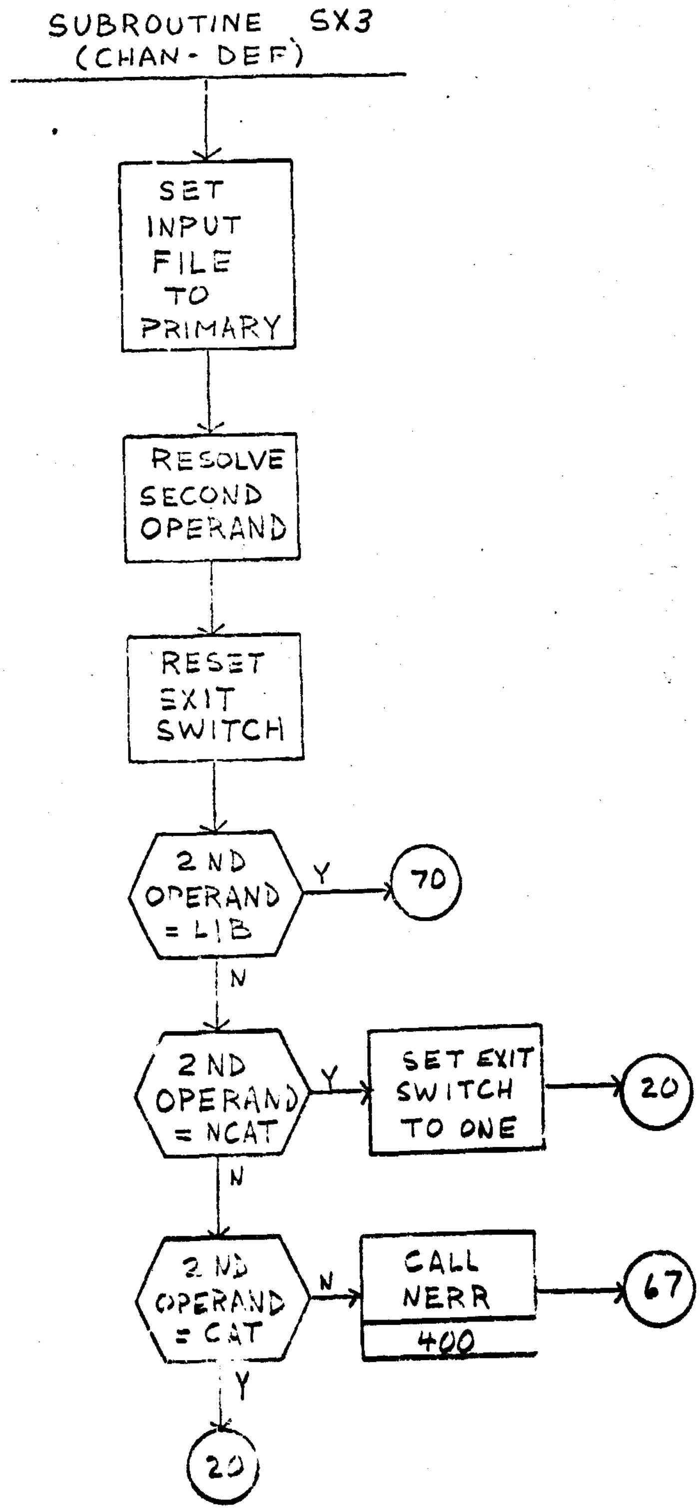

68
SUBTRACT ONE FROM IMEMD COUNT
RETURN

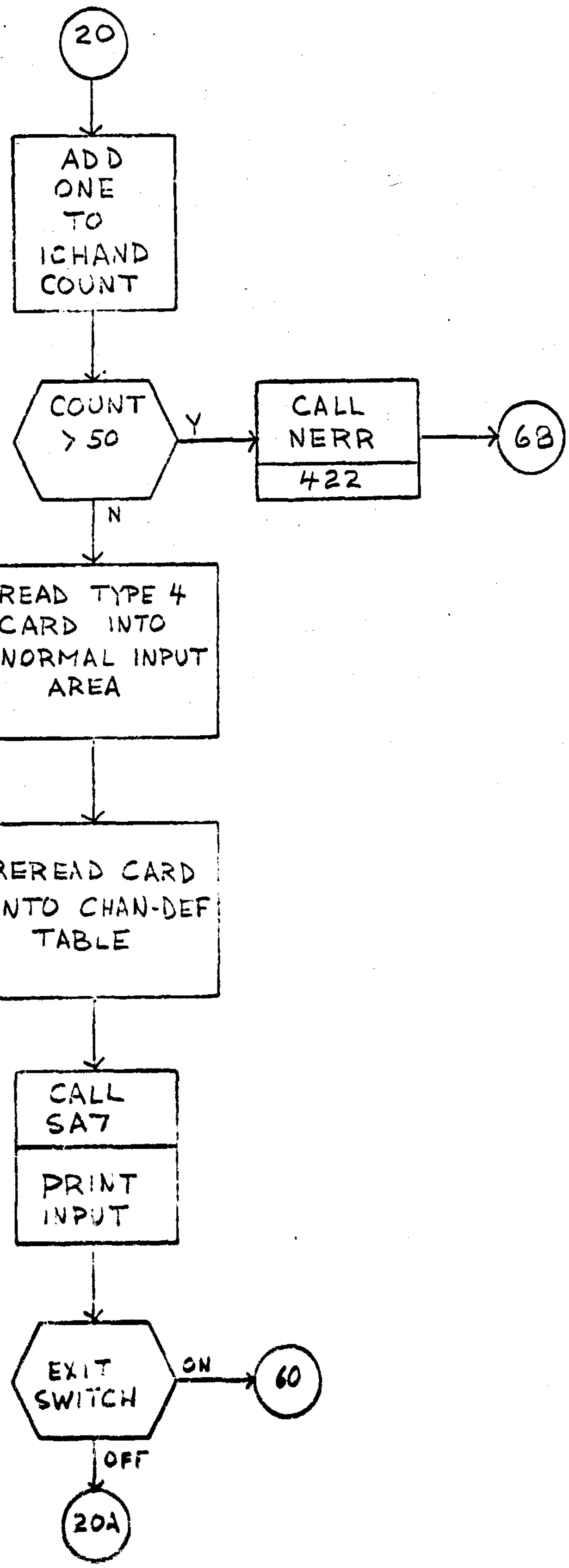

70
MOVE MEM-ID TO CUROUT AREA
CALL RBLK
RIGHT-FILL WITH BLANKS
CALL SRCHNV
FIND MEM-DEF
DEFINED
N
CALL NERR
414
68
Y
SKIP FIRST TWO WORDS IN MEM-DEF TABLE
70A


70A
CALL NXTIMG
READ RECORD INTO TABLE
ADJUST PAGE
PRINT MEM-END
65

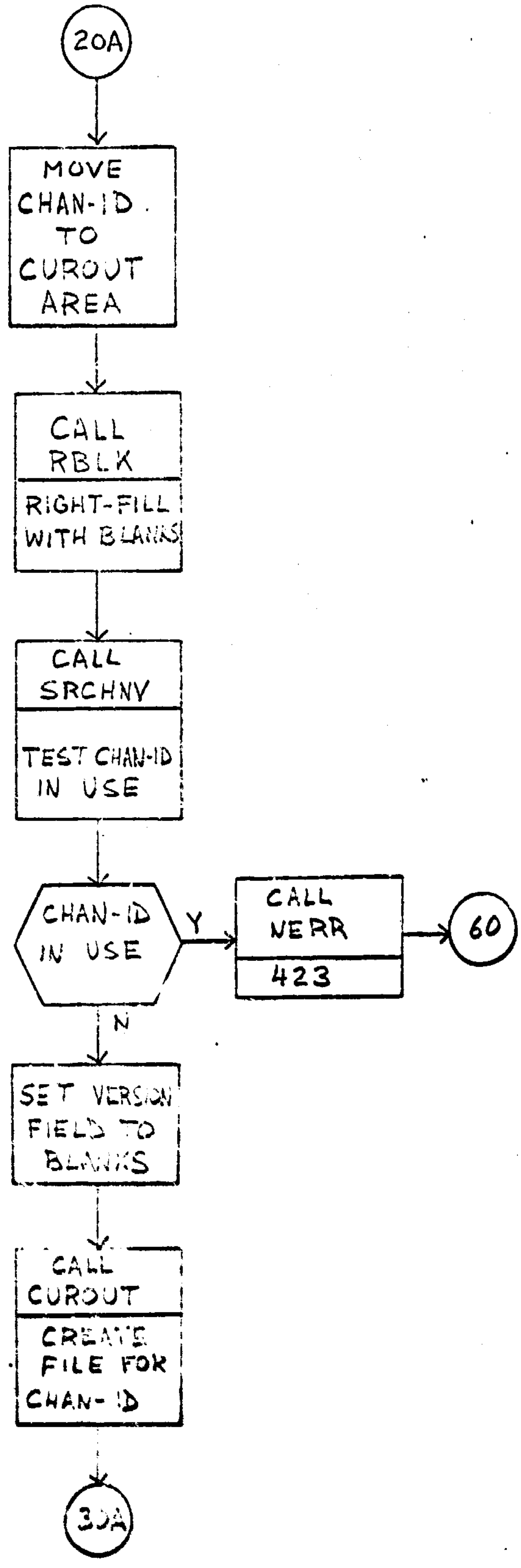
SUBROUTINE SX3

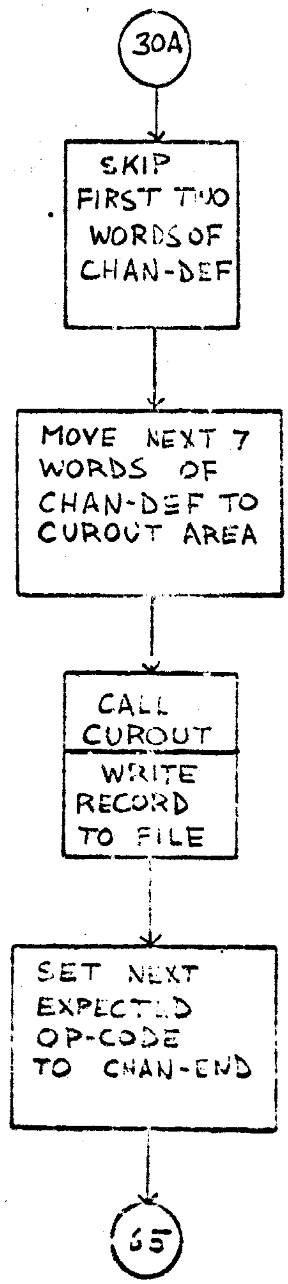
This routine processes channel definitions. The second operand is checked to see whether the definition is in the library or, if it is in the input stream, whether it is to be catalogued or not. The channel definition is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. A count is made of the number of channel definitions. A check is made to see if the number of channel definitions has exceeded the maximum permitted.

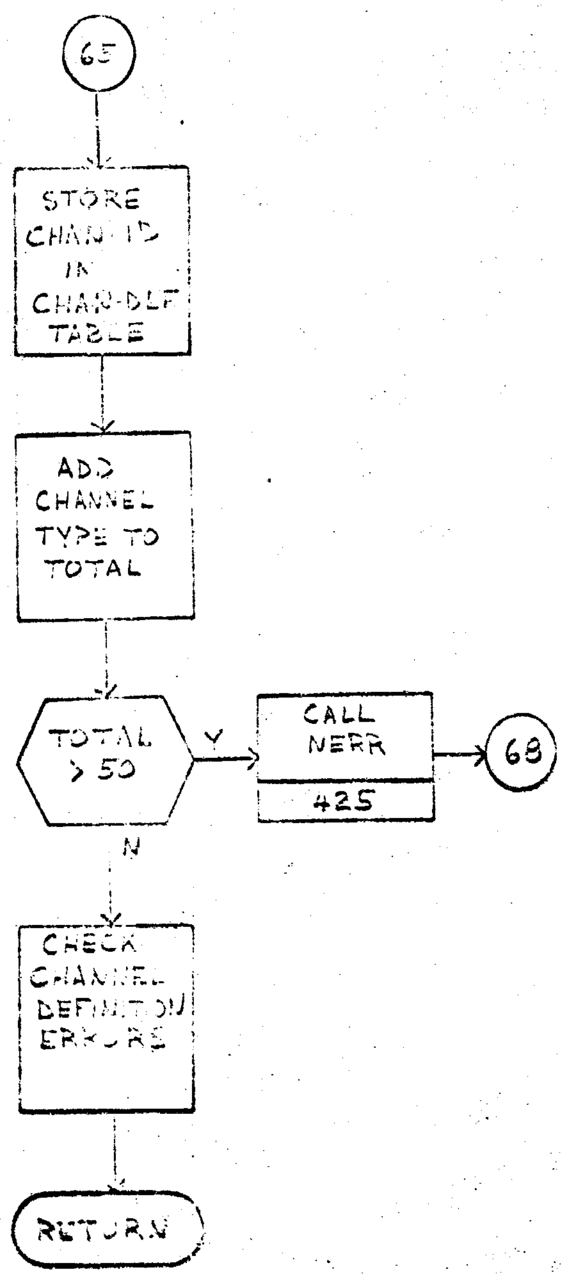

SUBROUTINE SX3
(CHAN-DEF)
SET INPUT FILE TO PRIMARY
RESOLVE SECOND OPERAND
RESET EXIT SWITCH
2ND OPERAND = LIB
Y
70
N
2ND OPERAND = NCAT
Y
SET EXIT SWITCH TO ONE
20
N
2ND OPERAND = CAT
N
CALL NERR
400
67
Y
20

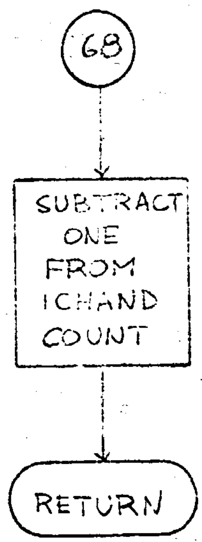


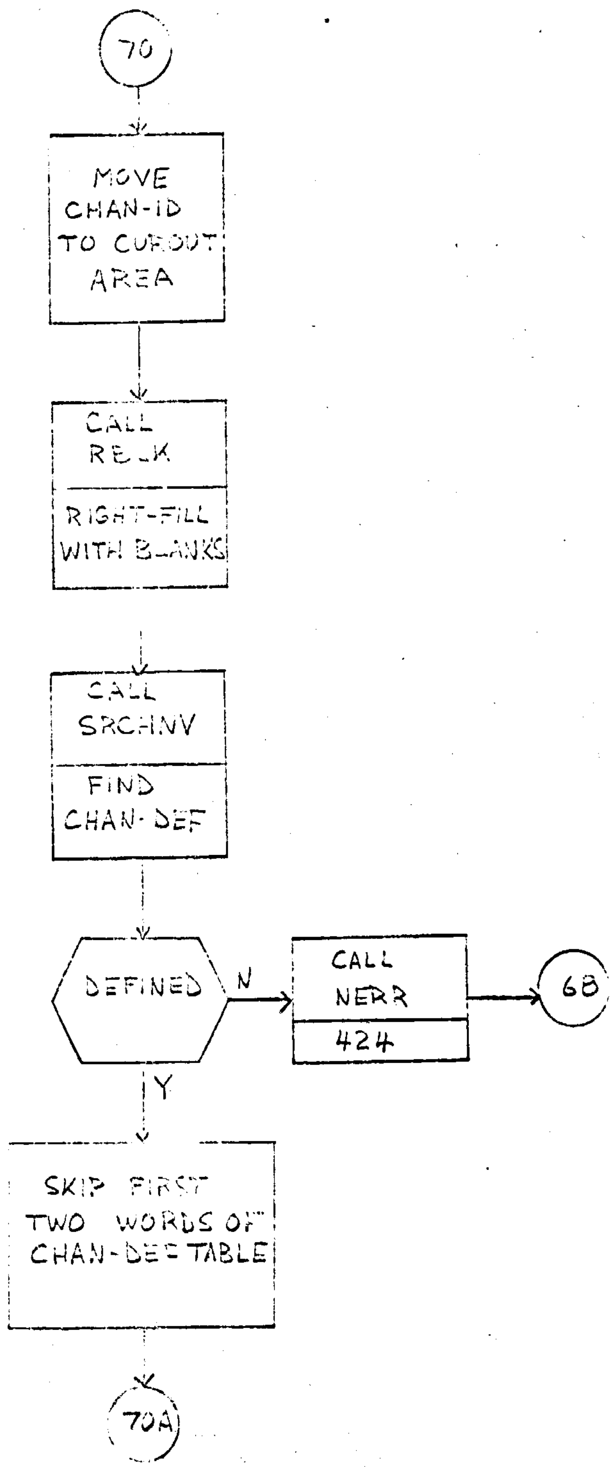
20
ADD ONE TO ICHAND COUNT
COUNT > 50
Y
CALL NERR
422
68
N
READ TYPE 4 CARD INTO NORMAL INPUT AREA
REREAD CARD INTO CHAN-DEF TABLE
CALL SA7
PRINT INPUT
EXIT SWITCH
ON
60
OFF
20A

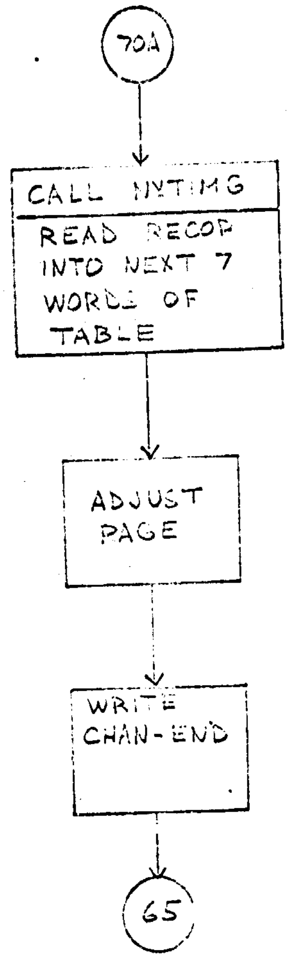

20A
MOVE CHAN-ID TO CUROUT AREA
CALL RBLK
RIGHT-FILL WITH BLANKS
CALL SRCHNV
TEST CHAN-ID IN USE
CHAN-ID IN USE
Y
CALL NERR
423
60
N
SET VERSION FIELD TO BLANKS
CALL CUROUT
CREATE FILE FOR CHAN-ID
30A


30A
SKIP FIRST TWO WORDS OF CHAN-DEF
MOVE NEXT 7 WORDS OF CHAN-DEF TO CUROUT AREA
CALL CUROUT
WRITE RECORD TO FILE
SET NEXT EXPECTED OP-CODE TO CHAN-END
65


65
STORE CHAN-ID IN CHAN-DEF TABLE
ADD CHANNEL TYPE TO TOTAL
TOTAL > 50
Y
N
CALL NERR
425
68
CHECK CHANNEL DEFINITION ERRORS
RETURN

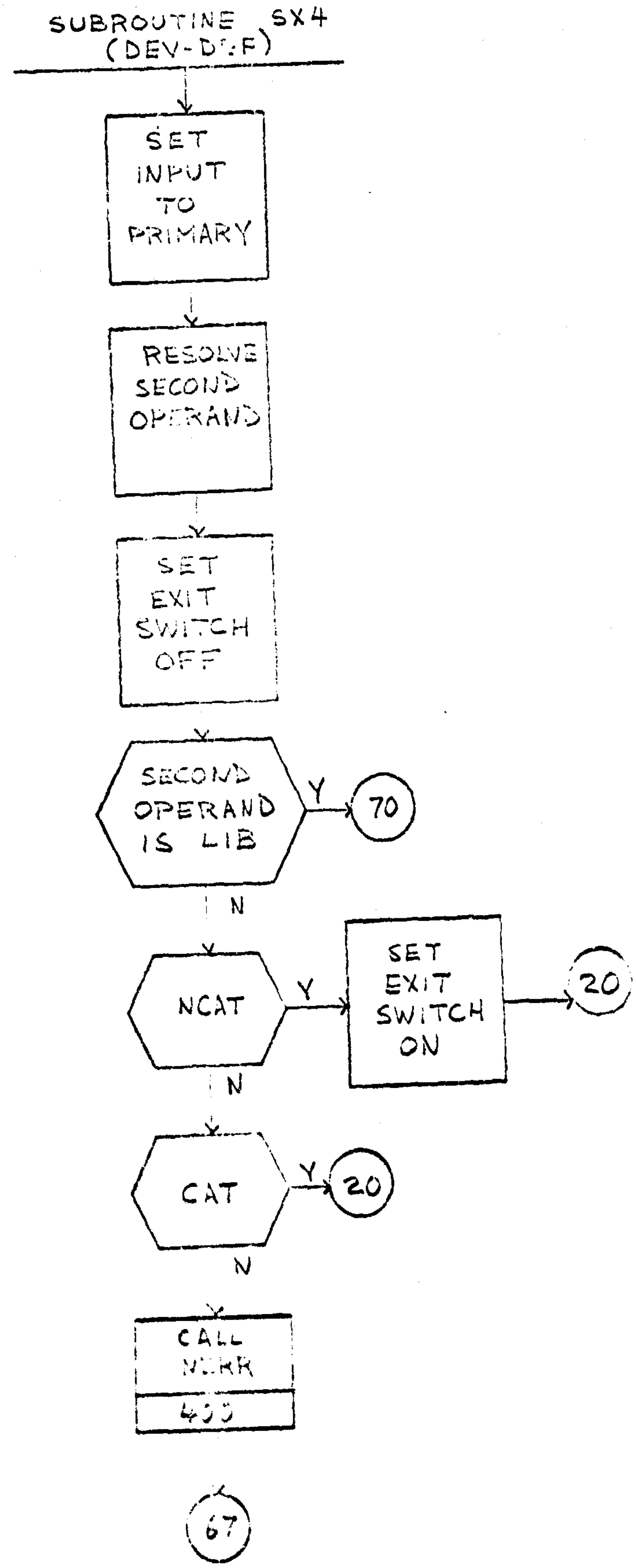

68
SUBTRACT ONE FROM ICHAND COUNT
RETURN

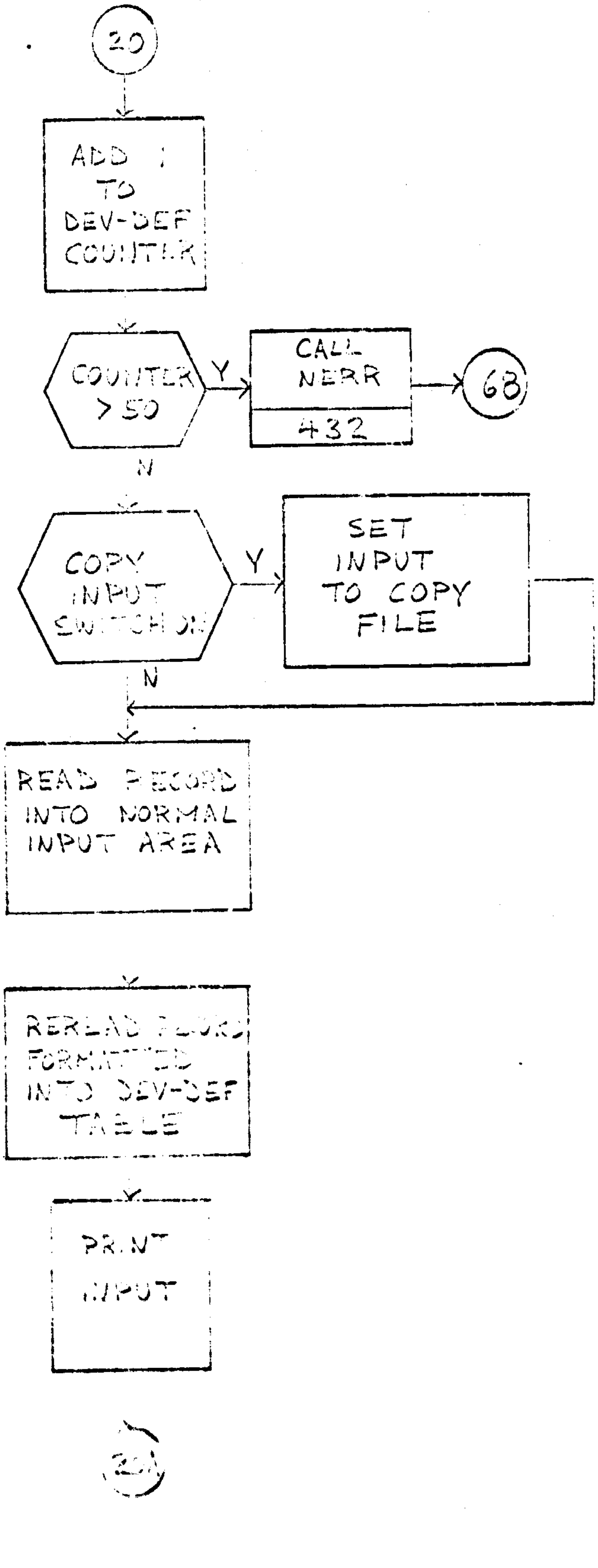

70
MOVE CHAN-ID TO CUROUT AREA
CALL RELK
RIGHT-FILL WITH BLANKS
CALL SRCHNV
FIND CHAN-DEF
DEFINED
N
CALL NERR
424
68
Y
SKIP FIRST TWO WORDS OF CHAN-DEF TABLE
70A

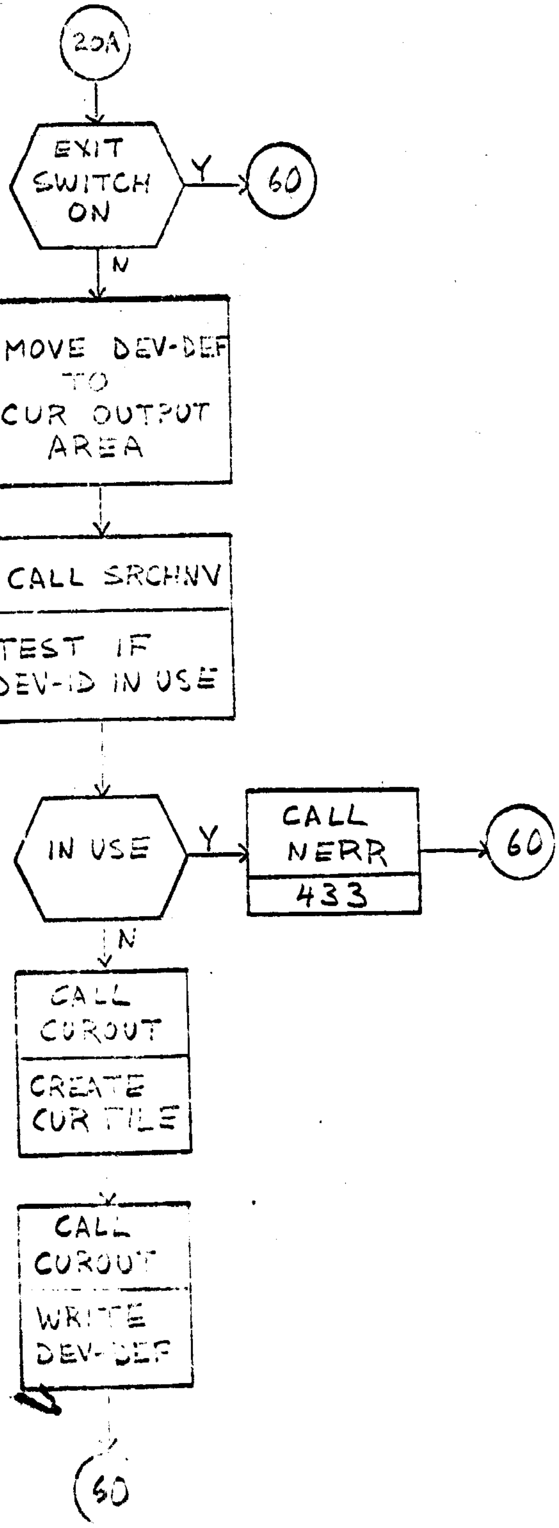

70A
CALL NYTIMG
READ RECORD INTO NEXT 7 WORDS OF TABLE
ADJUST PAGE
WRITE CHAN-END
65

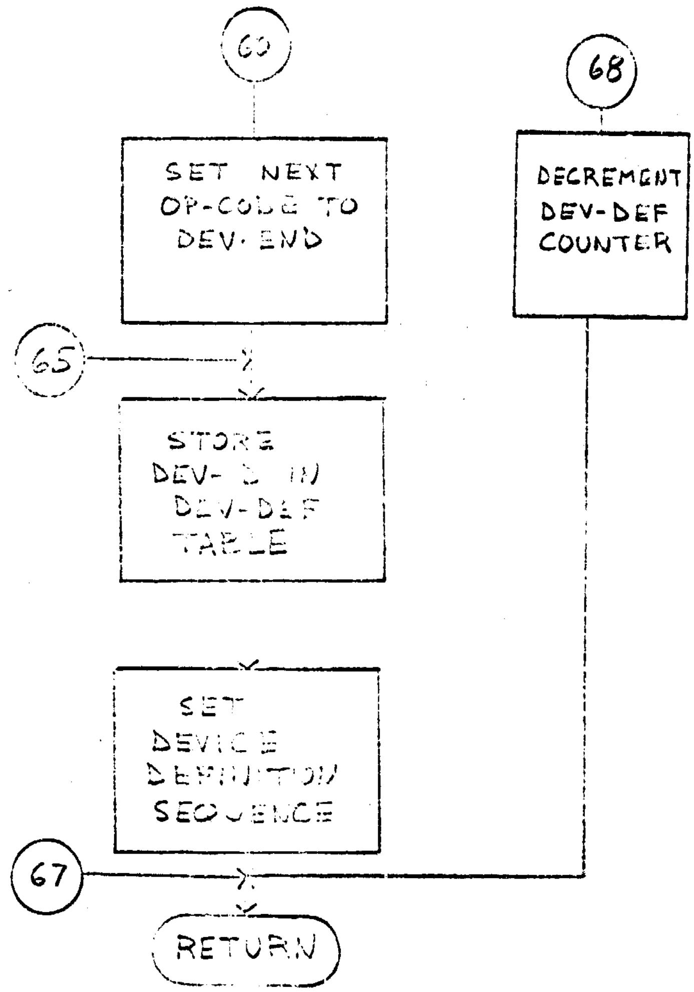
SUBROUTINE SX4

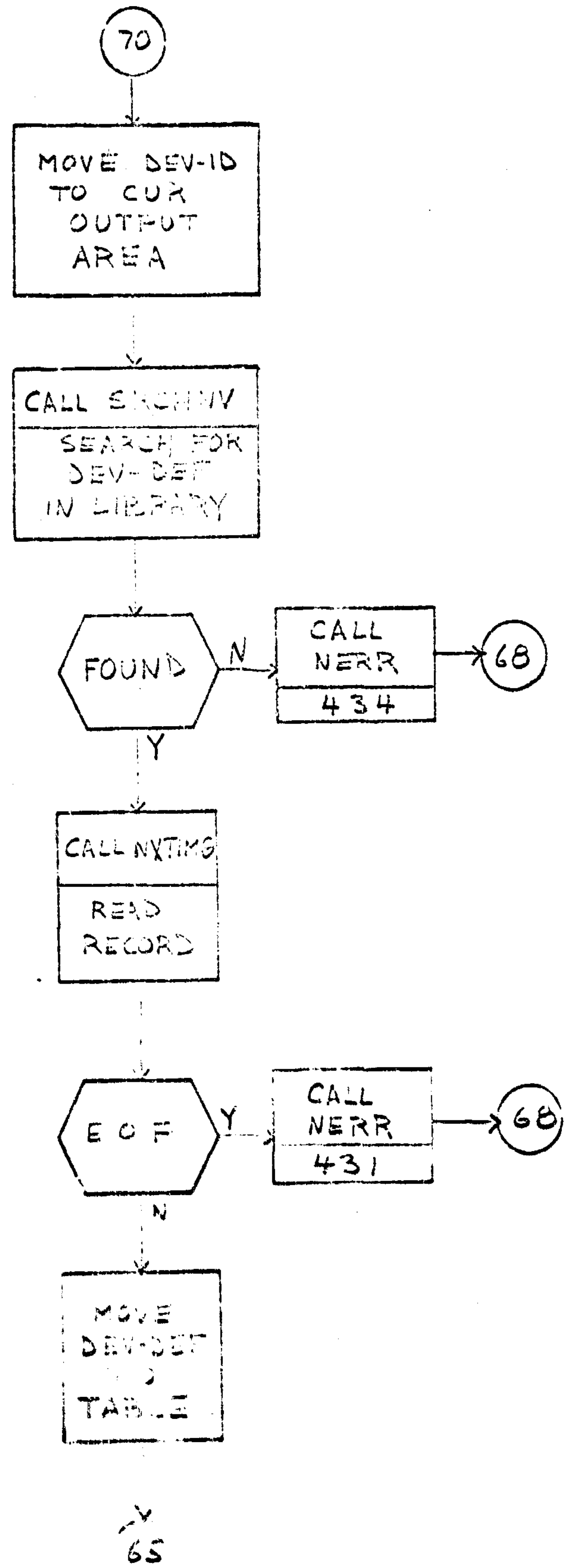
This routine processes a device definition statement. In this routine, the second operand is checked first to see if it indicates that the device definition is to be taken from the library, or if it is in the input stream, whether it is to be catalogued or not. The device definition is then read from the appropriate input file. It is then re-read into the device definition table. If the device defintion is to be catalogued, a check is made to see if the device ID is already in use. If it is, an error is written. If not, a new file is created for the device definition and the device defintion is catalogued.


SUBROUTINE SX4
(DEV-DEF)
SET INPUT TO PRIMARY
RESOLVE SECOND OPERAND
SET EXIT SWITCH OFF
SECOND OPERAND IS LIB
Y
70
N
NCAT
Y
SET EXIT SWITCH ON
20
N
CAT
Y
20
N
CALL NERR
400
67


20
ADD 1 TO DEV-DEF COUNTER
COUNTER > 50
Y
CALL NERR
432
68
N
COPY INPUT SWITCH ON
Y
SET INPUT TO COPY FILE
N
READ RECORD INTO NORMAL INPUT AREA
REREAD RECORD FORMATTED INTO DEV-DEF TABLE
PRINT INPUT
20A


20A
EXIT SWITCH ON
Y
60
N
MOVE DEV-DEF TO CUR OUTPUT AREA
CALL SRCHNV
TEST IF DEV-ID IN USE
IN USE
Y
CALL NERR
433
60
N
CALL CUROUT
CREATE CUR FILE
CALL CUROUT
WRITE DEV-DEF
60

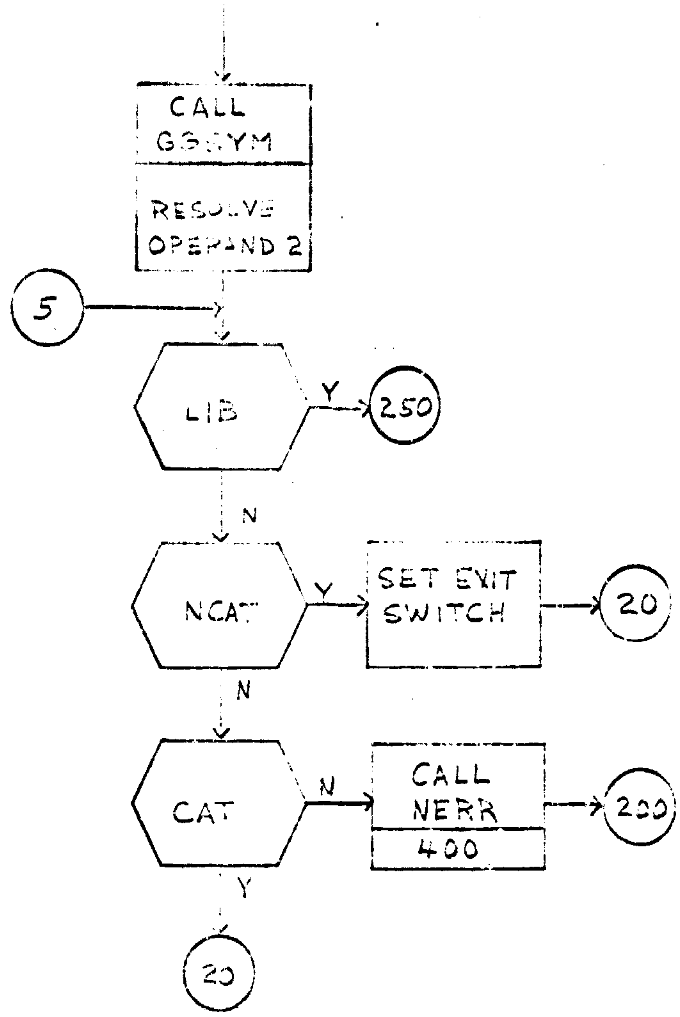

60
68
SET NEXT OP-CODE TO DEV-END
DECREMENT DEV-DEF COUNTER
65
STORE DEV-D IN DEV-DEF TABLE
SET DEVICE DEFINITION SEQUENCE
67
RETURN

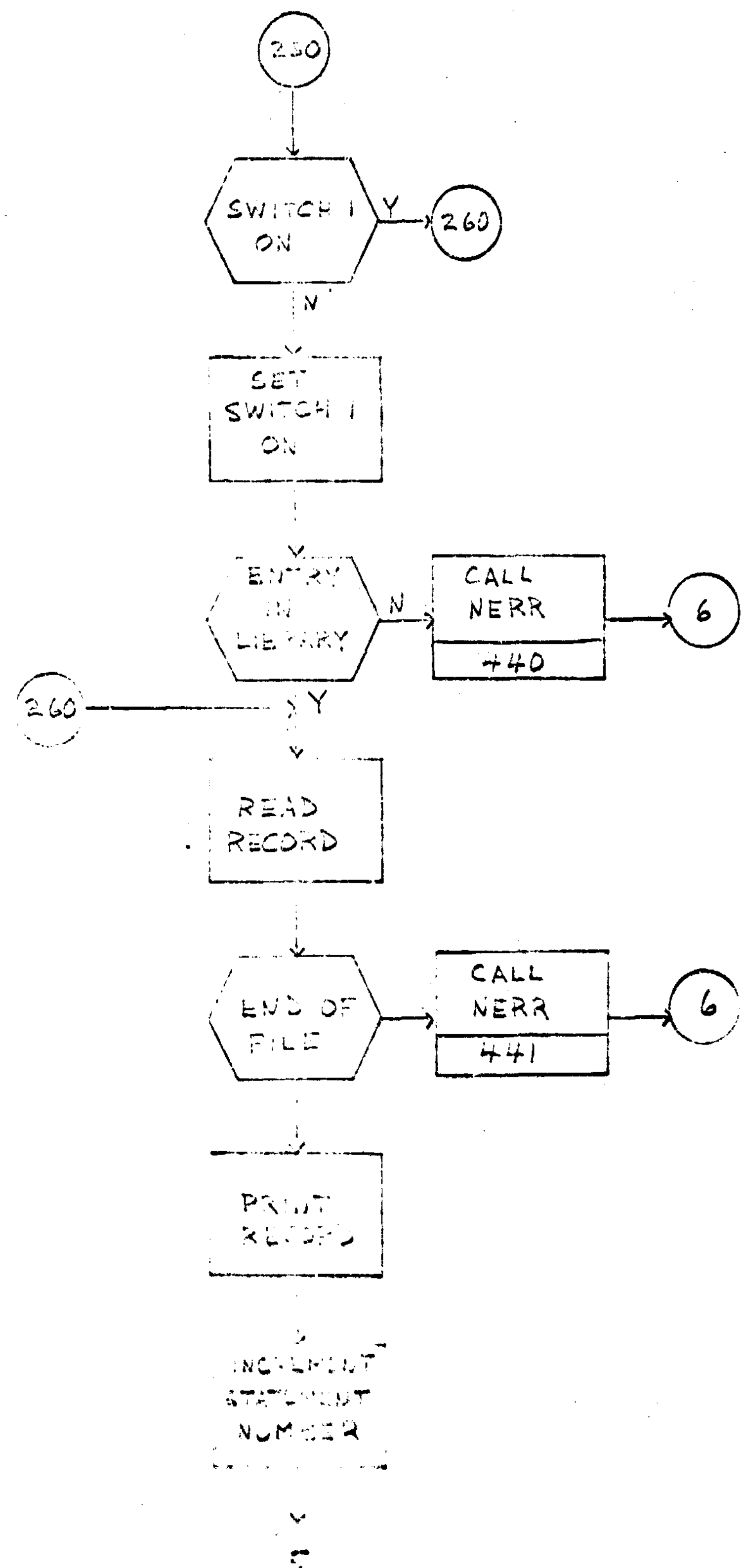

70
MOVE DEV-ID
TO CUR
OUTPUT
AREA
CALL
SEARCH FOR
DEV-DEF
IN LIBRARY
FOUND
N
Y
CALL
NERR
434
68
CALL NXTIMG
READ
RECORD
EOF
Y
N
CALL
NERR
431
68
MOVE
DEV-DEF
TABLE
65

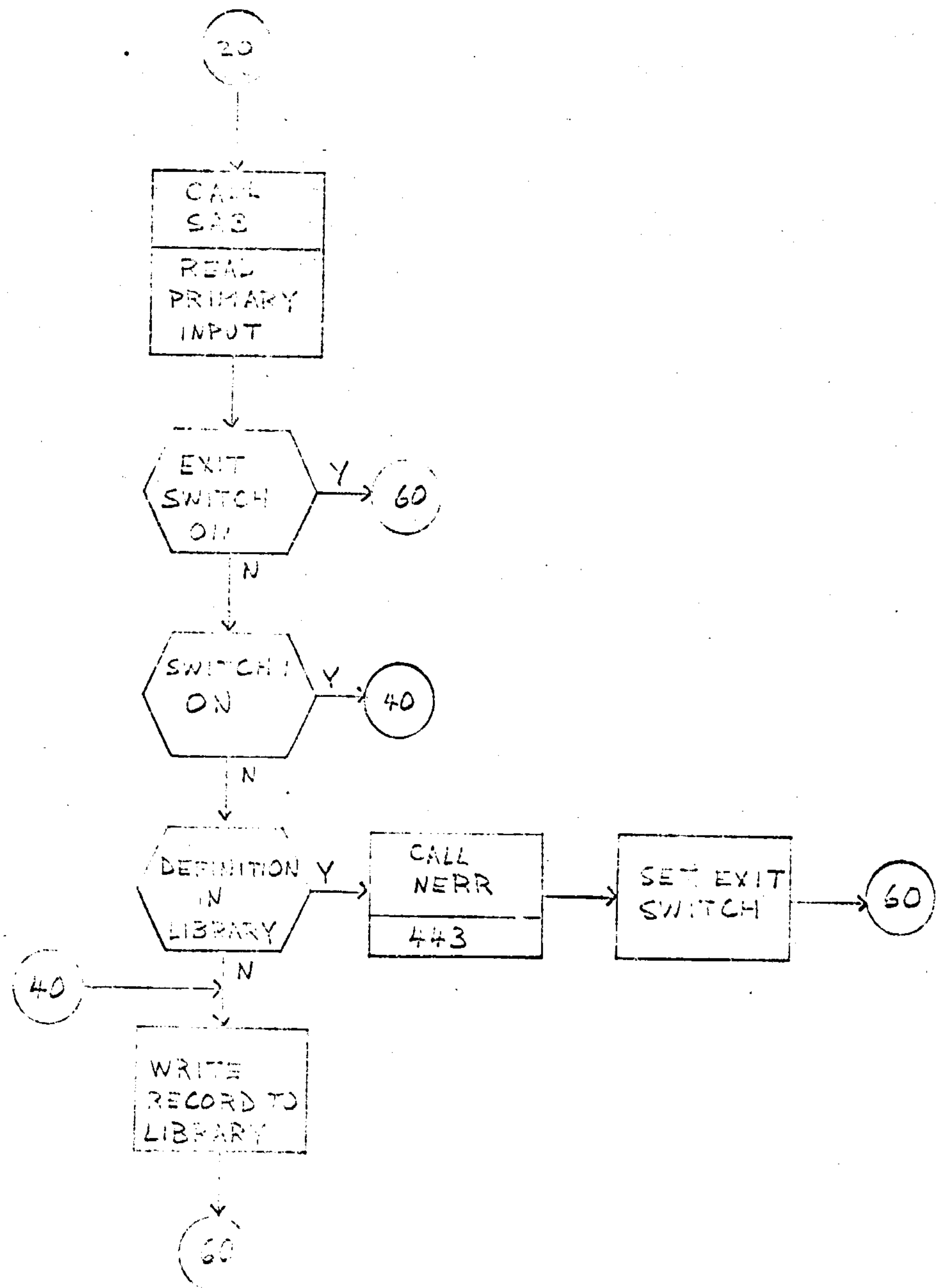
SUBROUTINE SX5

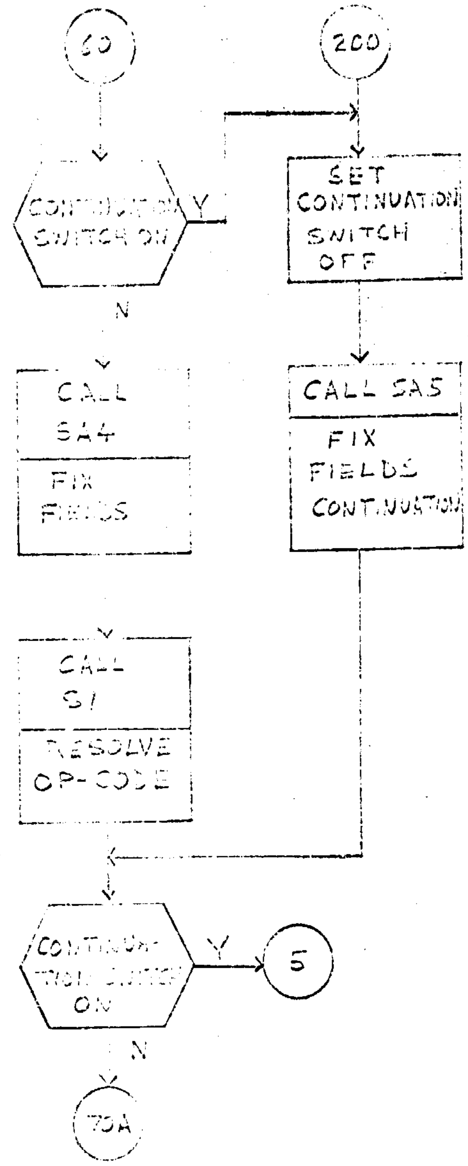
This routine processes the configuration card. The second operand is checked to see whether the configuration is in the library or, if it is in the input stream, whether it is to be catalogued or not. The next configuration card is then obtained from the appropriate input file, printed out, and stored in an appropriate table in common. As each subsequent input is read in, this routine calls other subroutines for further processing according to whether the input is for CPU, MEM, CHAN, CTL, or DEV configuration information. This routine also checks for the CONFIG-END card, at which point a return is made to the first pass subroutine, ASM1.

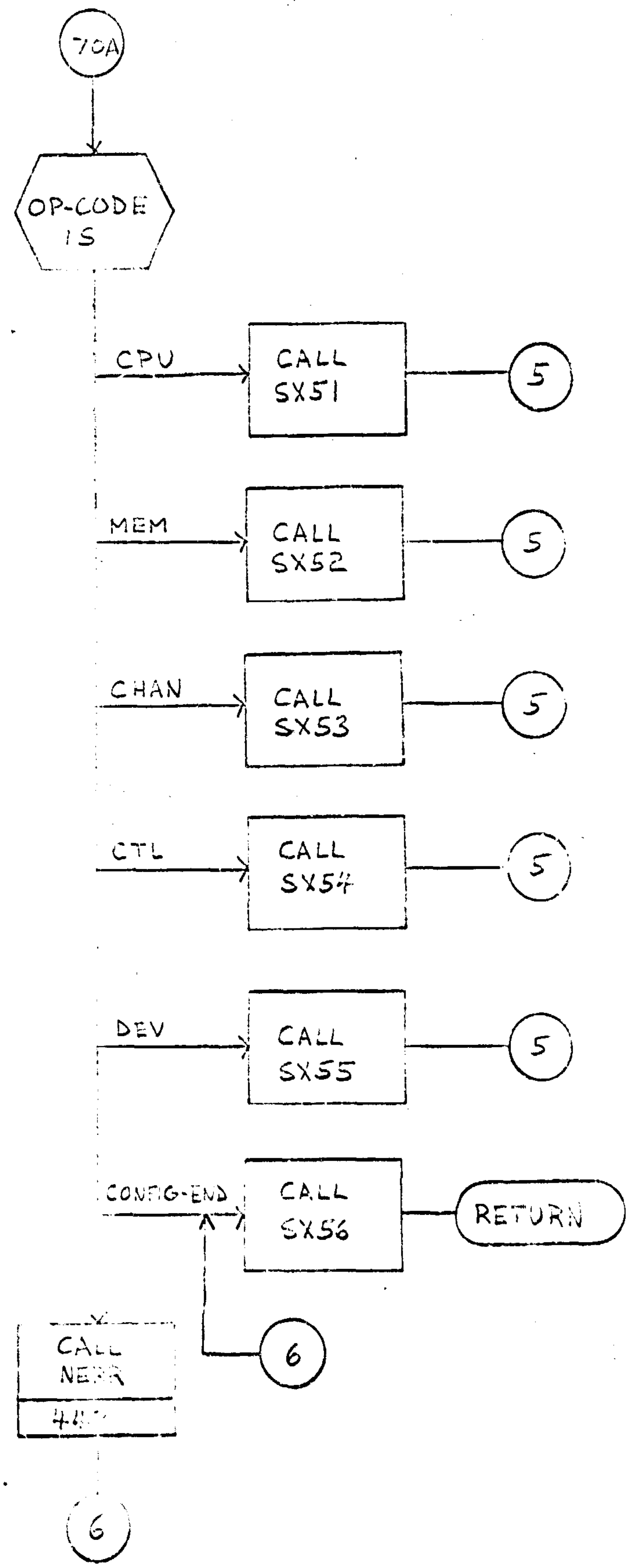
SUBROUTINE SX5
(CONFIG CARD)




250
SWITCH 1 ON
Y
260
N
SET SWITCH 1 ON
ENTRY IN LIBRARY
N
CALL NERR
440
6
260
Y
READ RECORD
END OF FILE
CALL NERR
441
6
PRINT RECORD
INCREMENT STATEMENT NUMBER
5


20
CALL SAB
READ PRIMARY INPUT
EXIT SWITCH ON
Y
60
N
SWITCH 1 ON
Y
40
N
DEFINITION IN LIBRARY
Y
CALL NERR
443
SET EXIT SWITCH
60
N
40
WRITE RECORD TO LIBRARY
60

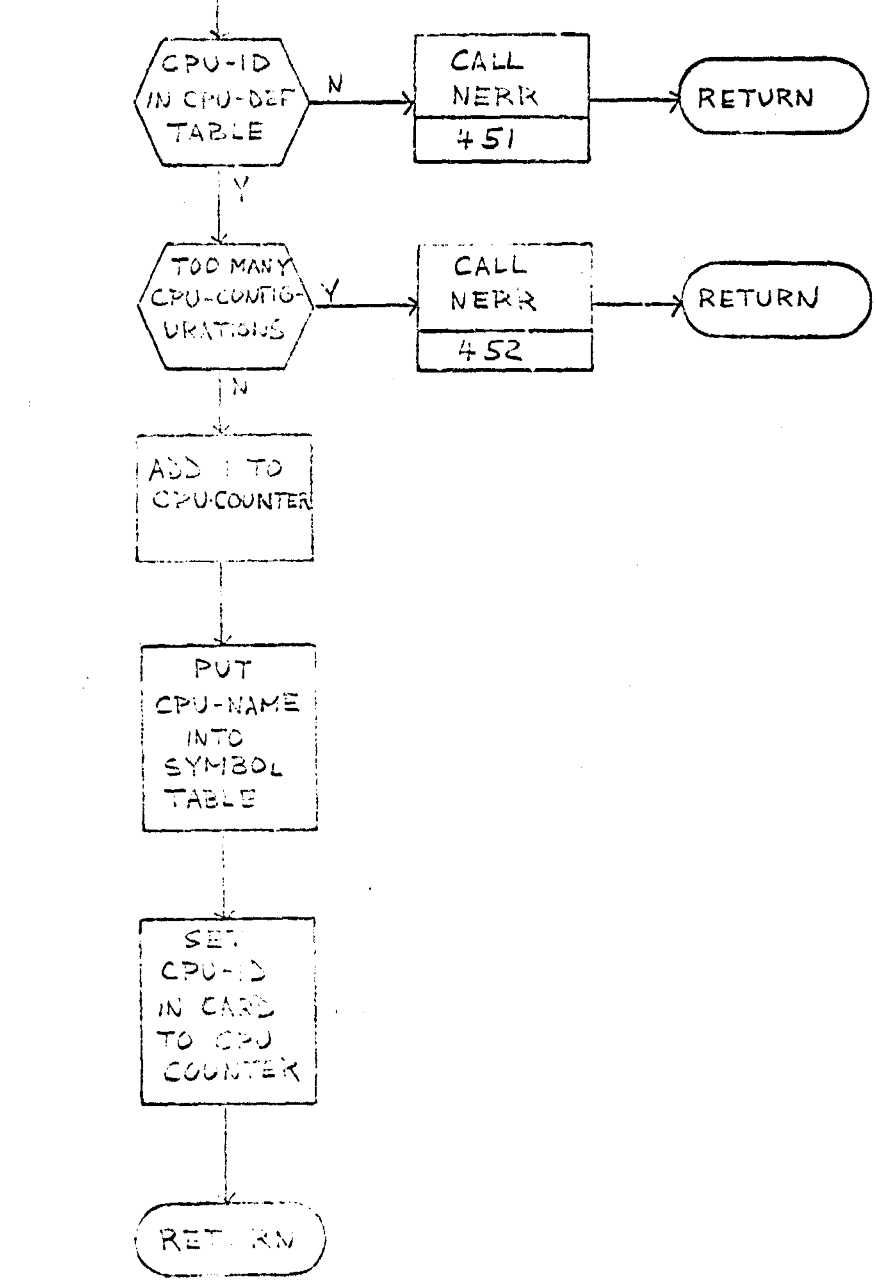

60
200
CONTINUATION SWITCH ON
Y
N
SET CONTINUATION SWITCH OFF
CALL SA4
FIX FIELDS
CALL SA5
FIX FIELDS CONTINUATION
CALL S1
RESOLVE OP-CODE
CONTINUATION SWITCH ON
Y
5
N
70A


70A
OP-CODE IS
CPU
CALL SX51
5
MEM
CALL SX52
5
CHAN
CALL SX53
5
CTL
CALL SX54
5
DEV
CALL SX55
5
CONFIG-END
CALL SX56
RETURN
6
6

SUBROUTINE SX51

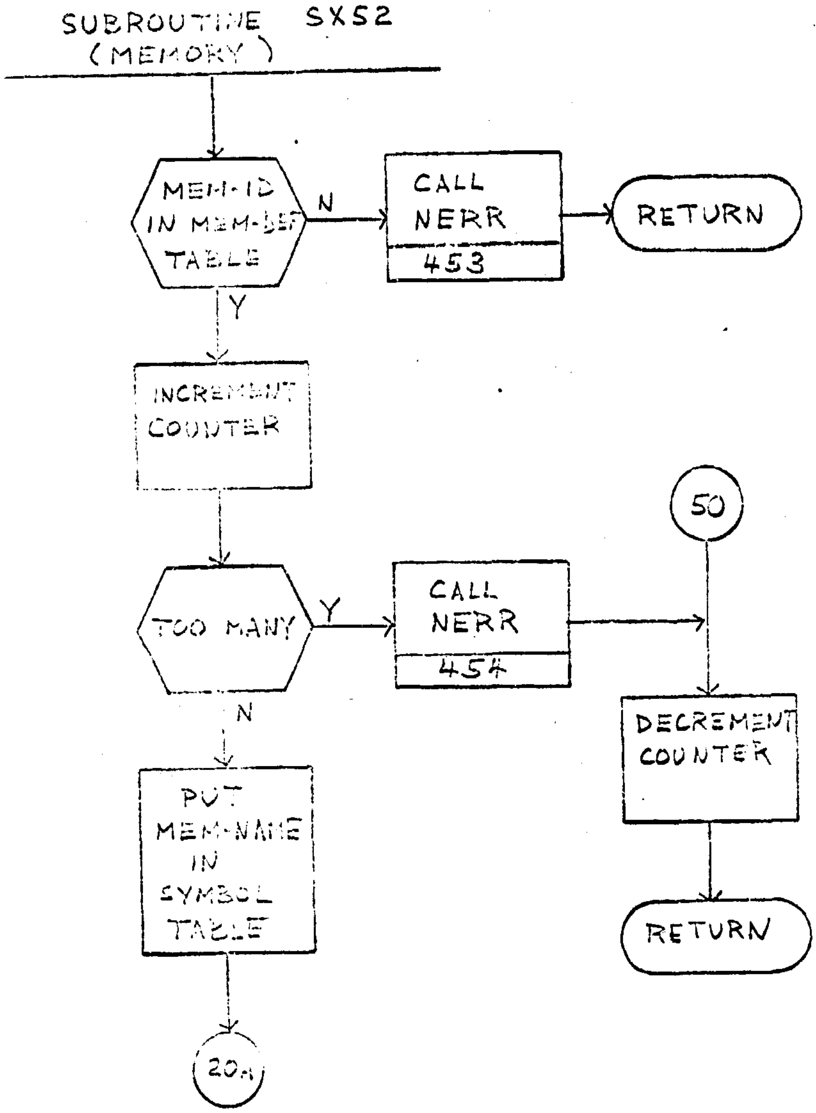
This routine processes the CPU configuration card. It first checks the CPU definition table to determine if the CPU-ID is valid. After finding the name, it is placed into the global symbol table. Some additional housekeeping is performed.

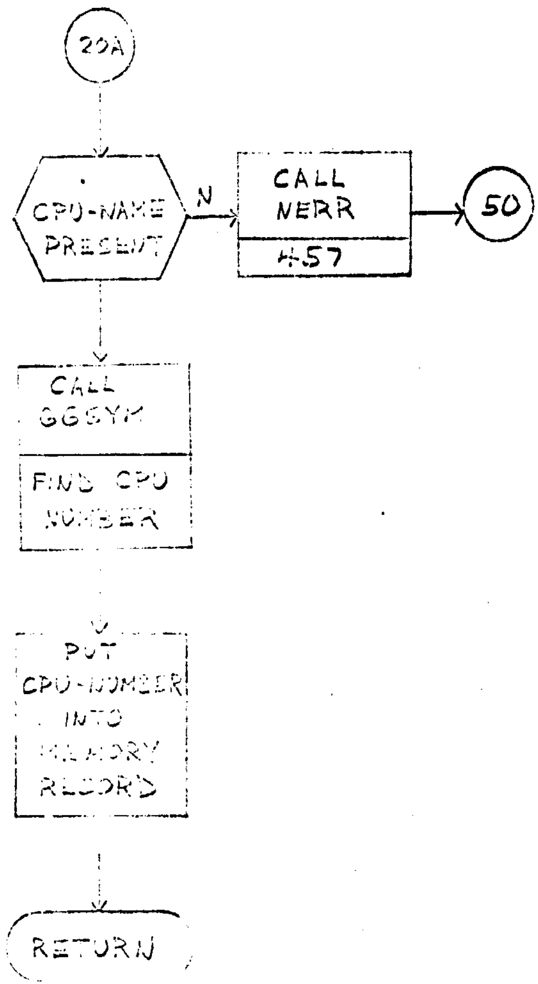
SUBROUTINE SX51
(CPU CONFIGURATION)



SUBROUTINE SX52

This routine processes the memory configuration card. The MEMORY-ID table is searched to verify that the MEMORY-ID is valid. A test is made to see that the maximum number of memory names is not exceeded. A test is made to see if the associated CPU name is valid.

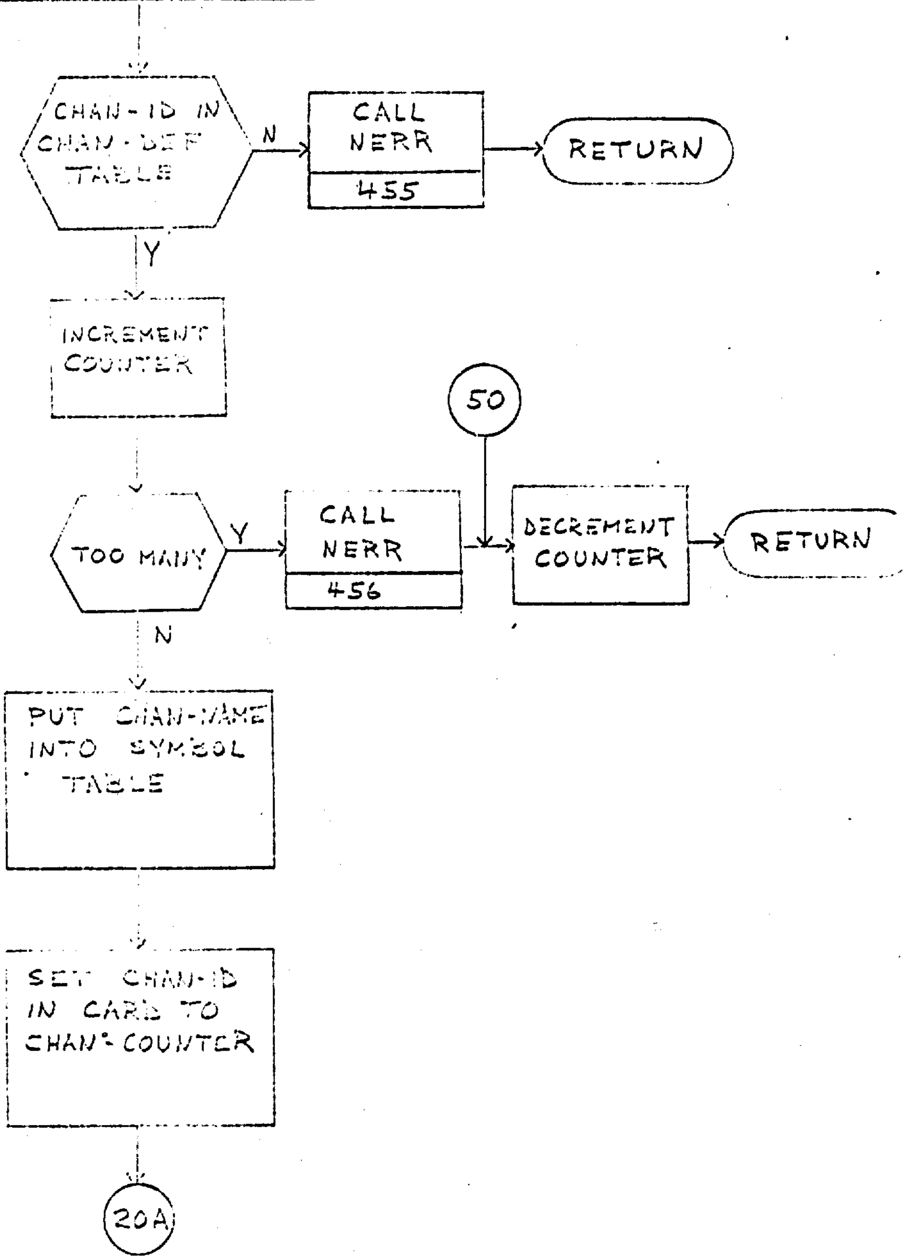

SUBROUTINE SXS2
(MEMORY)
MEM-ID IN MEM-DEF TABLE
N
Y
CALL NERR
453
RETURN
INCREMENT COUNTER
50
TOO MANY
Y
N
CALL NERR
454
DECREMENT COUNTER
PUT MEM-NAME IN SYMBOL TABLE
RETURN
20A

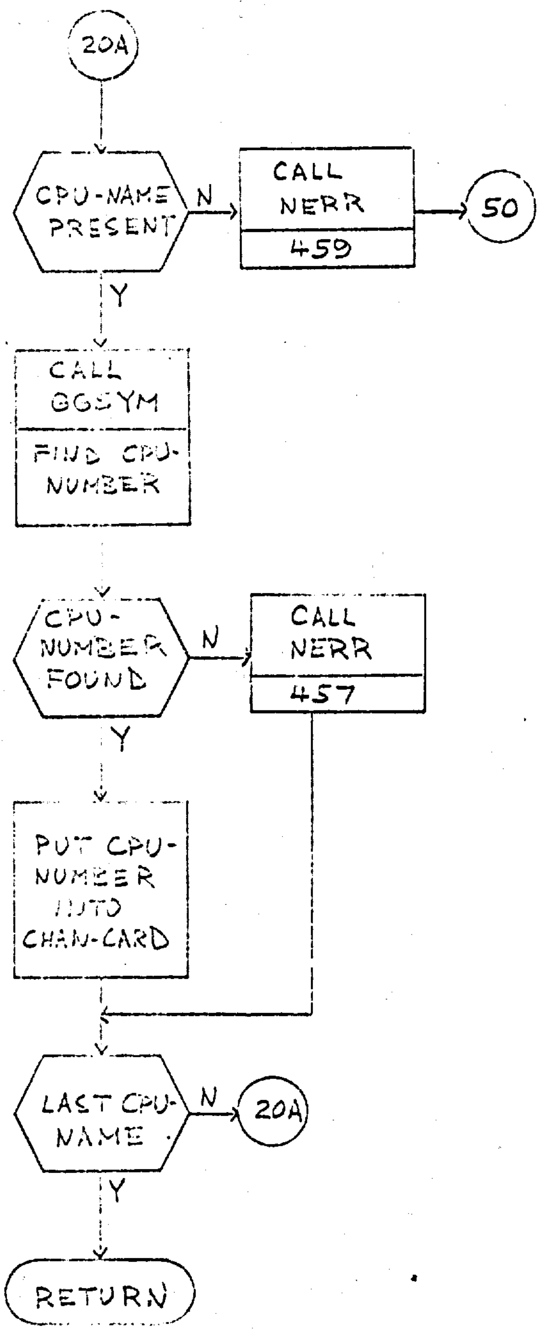

20A
CPU-NAME PRESENT
N
CALL NERR
457
50
CALL GGSYM
FIND CPU NUMBER
PUT CPU-NUMBER INTO MEMORY RECORD
RETURN

SUBROUTINE SX53

This routine processes the channel configuration card. The CHANNEL-ID table is searched to verify that the CHANNEL-ID is valid. A test is made to see that the maximum number of channel names is not exceeded. A test is made to see if the associated CPU name is valid.

SUBROUTINE SX53
(CHANNEL)

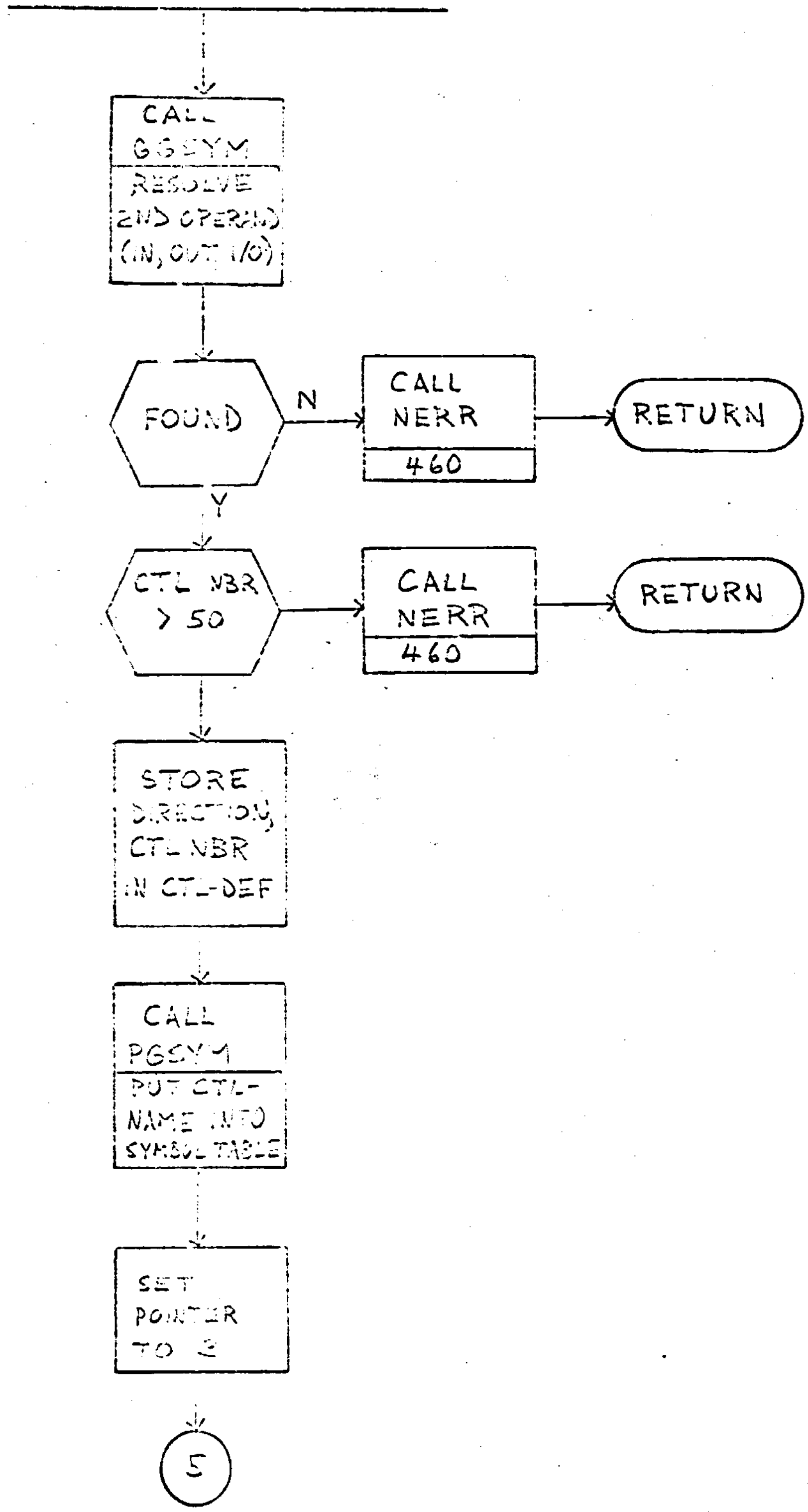


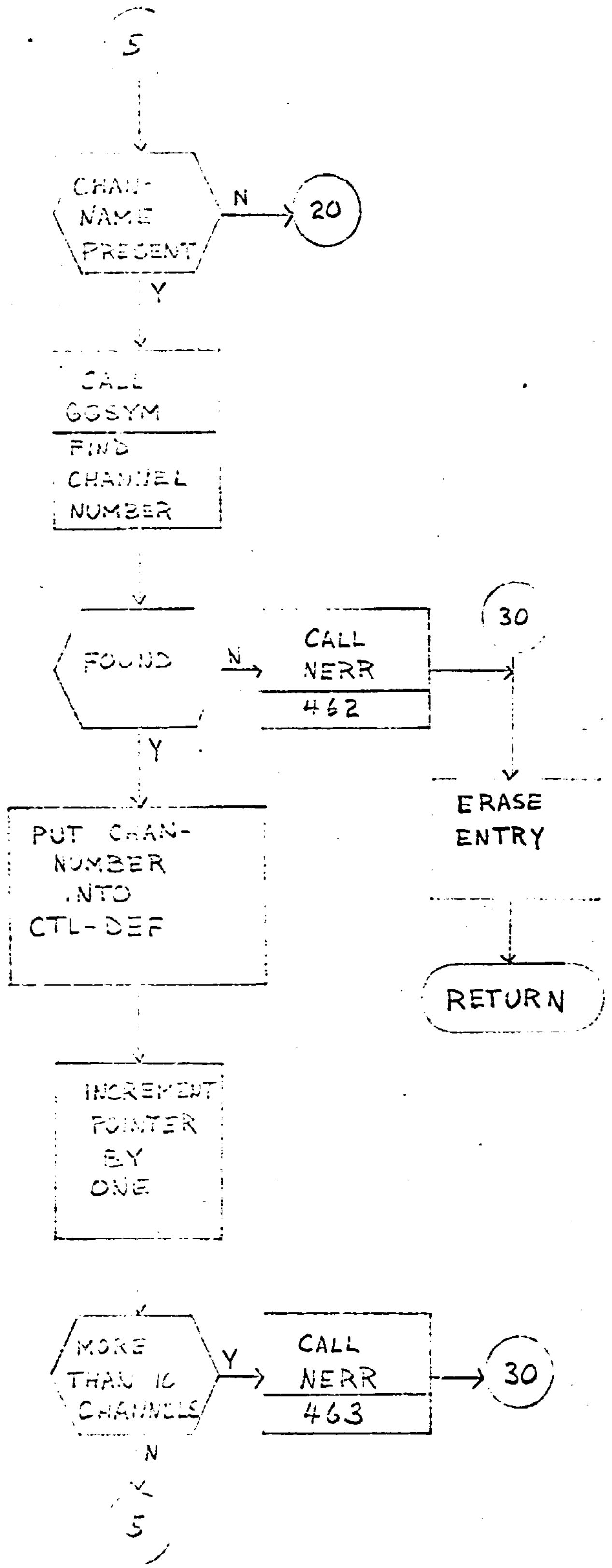

20A
CPU-NAME PRESENT
N
CALL NERR
459
50
Y
CALL GGSYM
FIND CPU-NUMBER
CPU-NUMBER FOUND
N
CALL NERR
457
Y
PUT CPU-NUMBER INTO CHAN-CARD
LAST CPU-NAME
N
20A
Y
RETURN

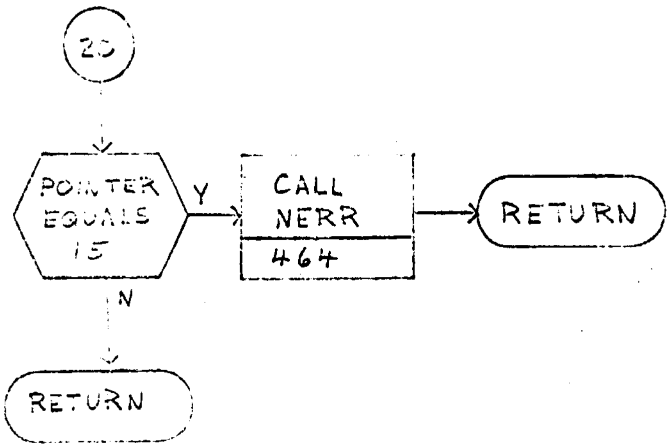
SUBROUTINE SX54

This routine processes the control configuration card. The CTL-ID table is searched to verify that the CTL-ID is valid. A test is made to see that the maximum number of control names is not exceeded. A test is made to see if the associated CPU name is valid.

SUBROUTINE SX54
(CONTROL)

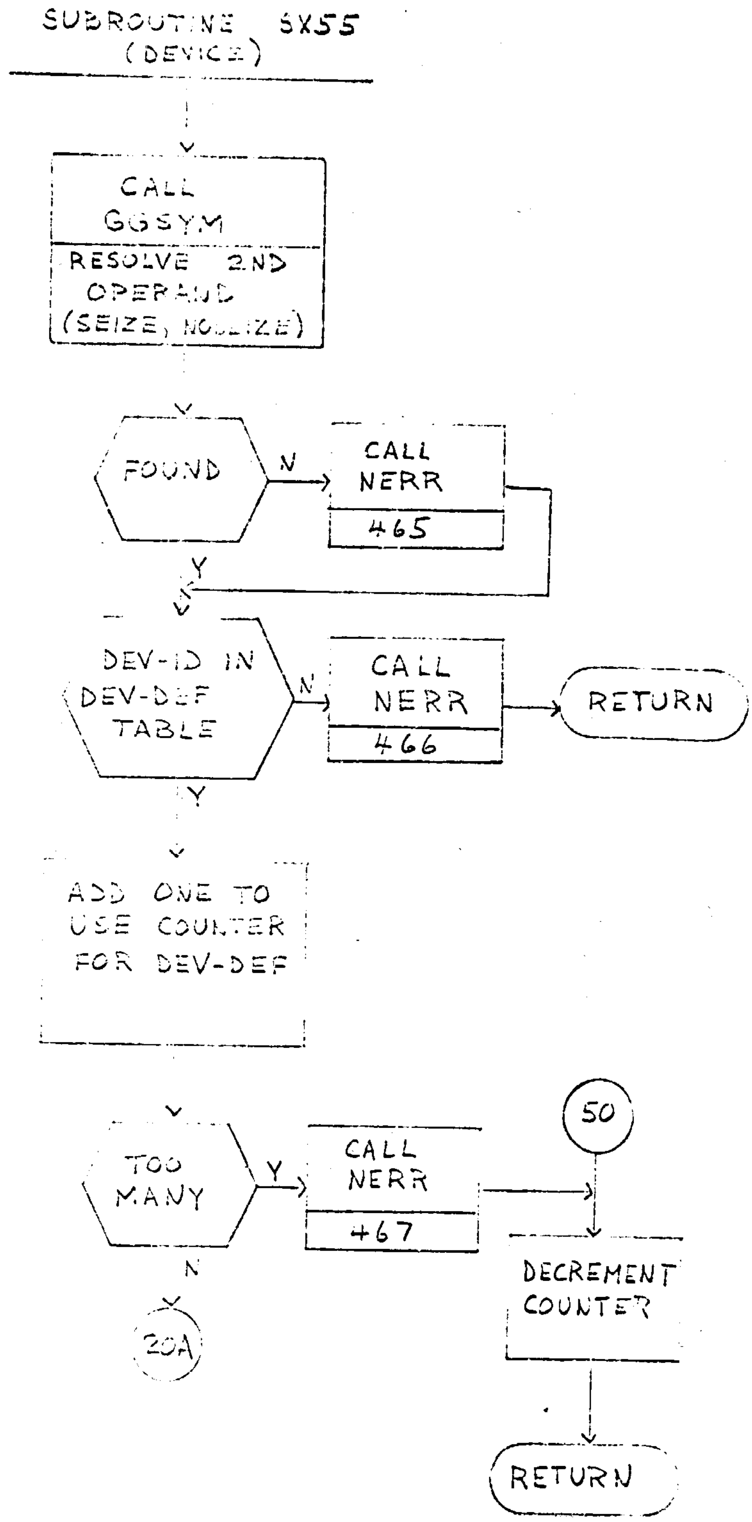


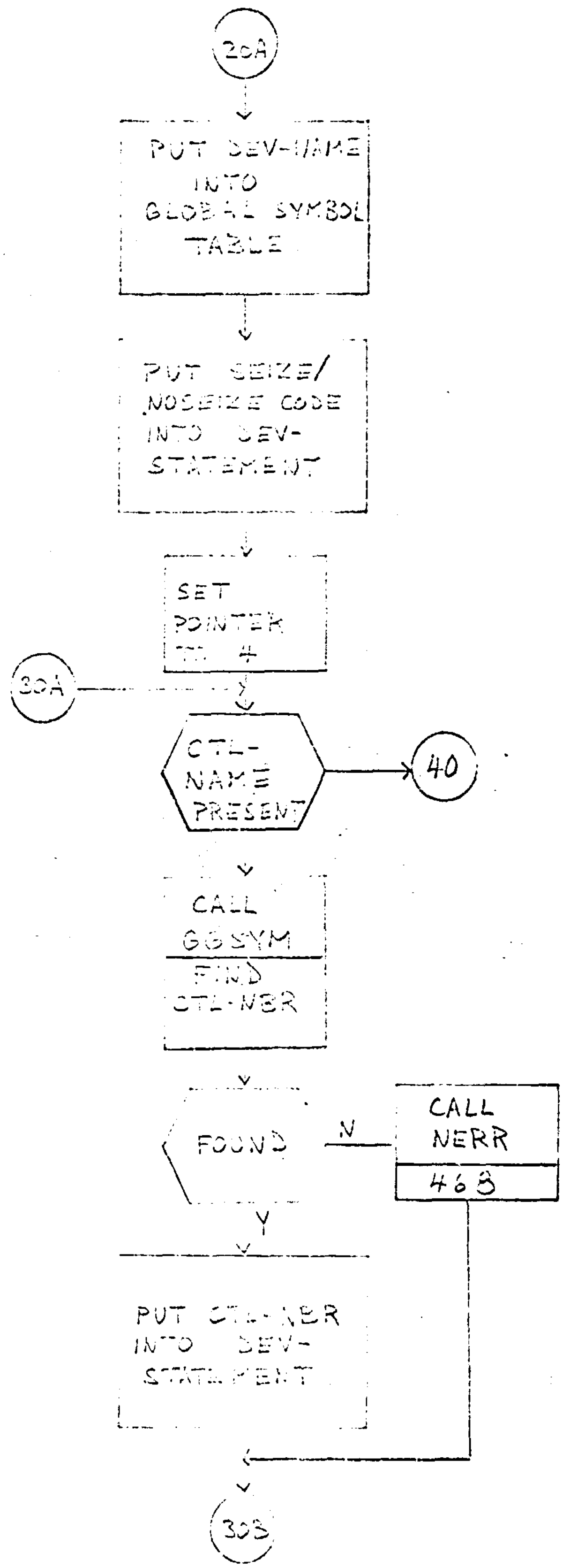

5
CHAN-NAME PRESENT
N
20
Y
CALL GOSYM
FIND CHANNEL NUMBER
FOUND
N
CALL NERR
462
30
Y
PUT CHAN-NUMBER INTO CTL-DEF
ERASE ENTRY
RETURN
INCREMENT POINTER BY ONE
MORE THAN 16 CHANNELS
Y
CALL NERR
463
30
N
5

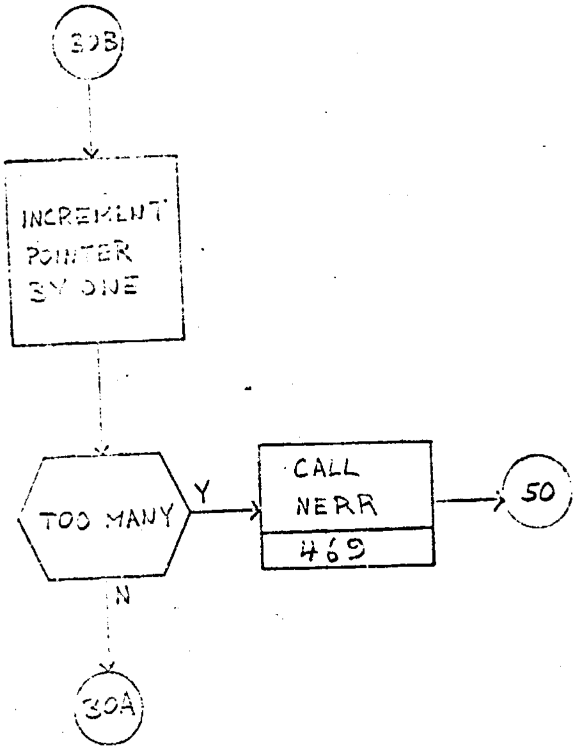

20
POINTER EQUALS 15
Y
N
CALL NERR
464
RETURN
RETURN

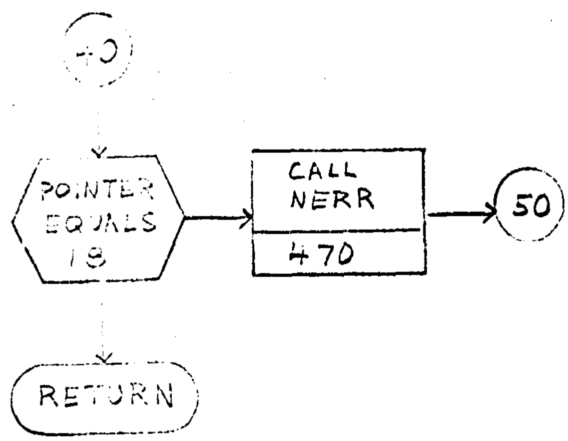
SUBROUTINE SX55

This routine processes the device configuration card. The first step is to resolve the SEIZE, NOSEIZE operand. A check is then made to verify that the device definition is valid. A check is made to see that the number of device definitions does not exceed the maximum. The device name is then placed in the global symbol table. The control names are resolved and appropriate housekeeping is performed.


SUBROUTINE SX55
(DEVICE)
CALL GGSYM
RESOLVE 2ND OPERAND (SEIZE, NOSEIZE)
FOUND
N
CALL NERR
465
Y
DEV-ID IN DEV-DEF TABLE
N
CALL NERR
466
RETURN
Y
ADD ONE TO USE COUNTER FOR DEV-DEF
TOO MANY
Y
CALL NERR
467
50
DECREMENT COUNTER
RETURN
N
20A


20A
PUT DEV-NAME INTO GLOBAL SYMBOL TABLE
PUT SEIZE/NOSEIZE CODE INTO DEV-STATEMENT
SET POINTER TO 4
30A
CTL-NAME PRESENT
40
CALL GGSYM
FIND CTL-NBR
FOUND
N
Y
CALL NERR
46B
PUT CTL-NBR INTO DEV-STATEMENT
30B

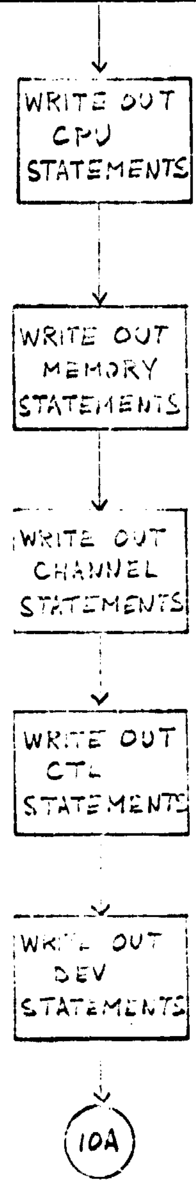

30B
INCREMENT
POINTER
BY ONE
TOO MANY
Y
N
CALL
NERR
469
50
30A


40
POINTER
EQUALS
18
CALL
NERR
470
50
RETURN

SUBROUTINE SX56

This routine processes the CONFIG-END card. Upon receiving this card, the hardware configuration tables built up by other subroutines are written out onto the simulator input tape. A return is then made to the first pass routine.

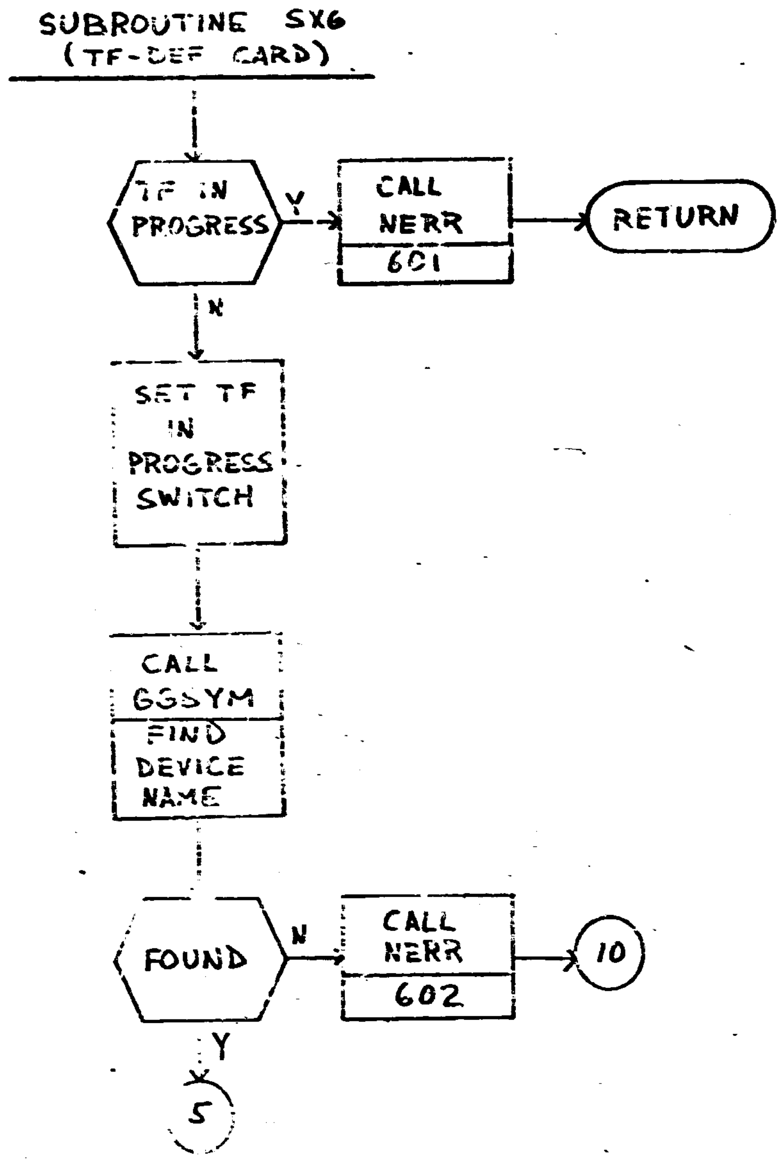
SUBROUTINE SX56
(CONFIG-END CARD)

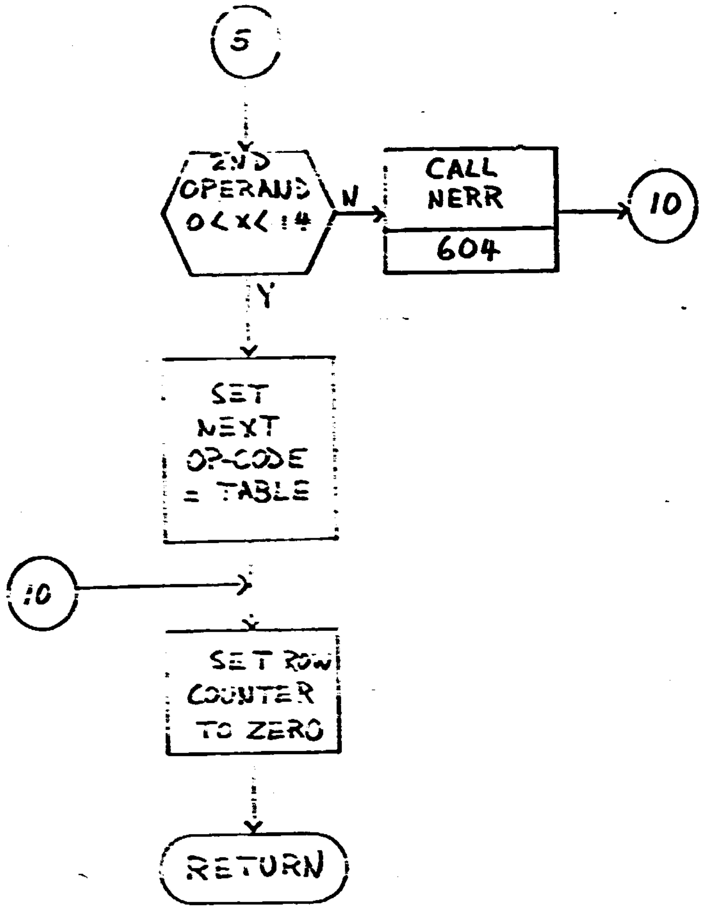



IOA
LIST CPU NAMES AND NUMBERS
LIST MEMORY NAMES AND NUMBERS
LIST CHANNEL NAMES AND NUMBERS
LIST CONTROL NAMES AND NUMBERS
LIST DEVICE NAMES AND NUMBERS
RETURN

SUBROUTINE SX6

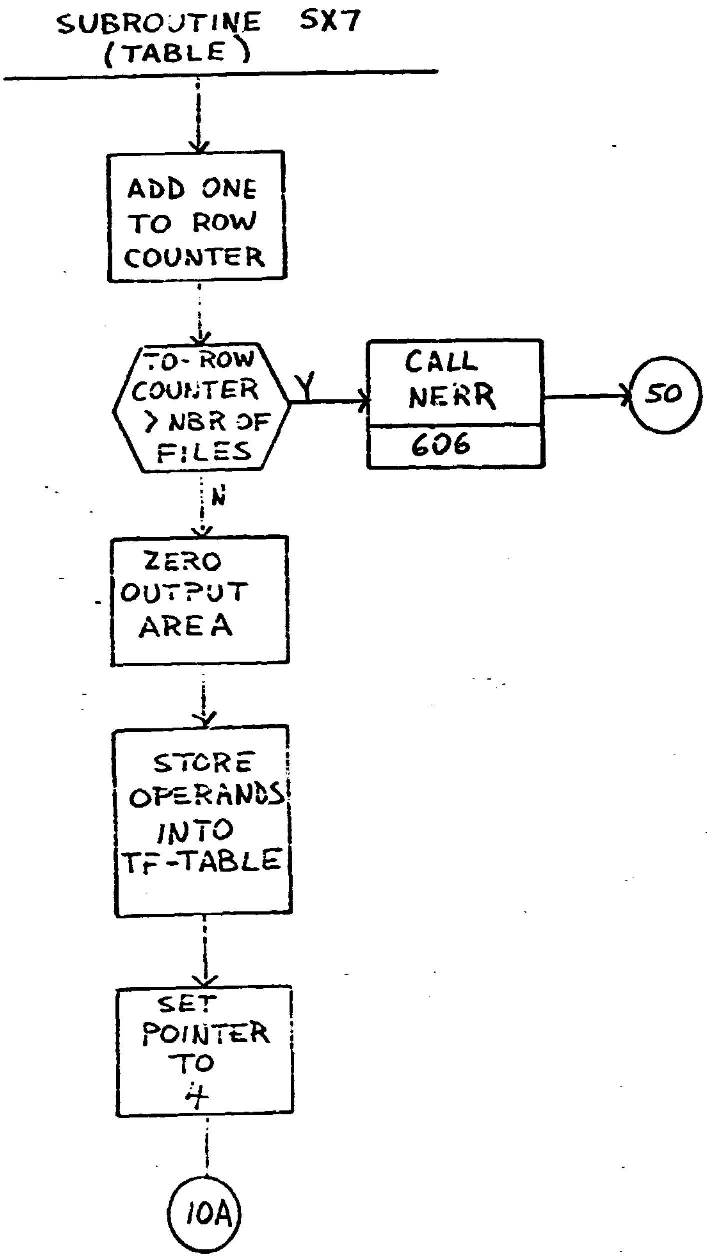
This routine processes the TF definition card. The first step is to test whether a TF definition is already in progress. If it is, an error is declared. If not, a TF definition is declared to be in progress. A check is then made for the device name and number of files. The next anticipated op-code is set to be TABLE. The row counter is then zeroed, and a return is made to the calling program.

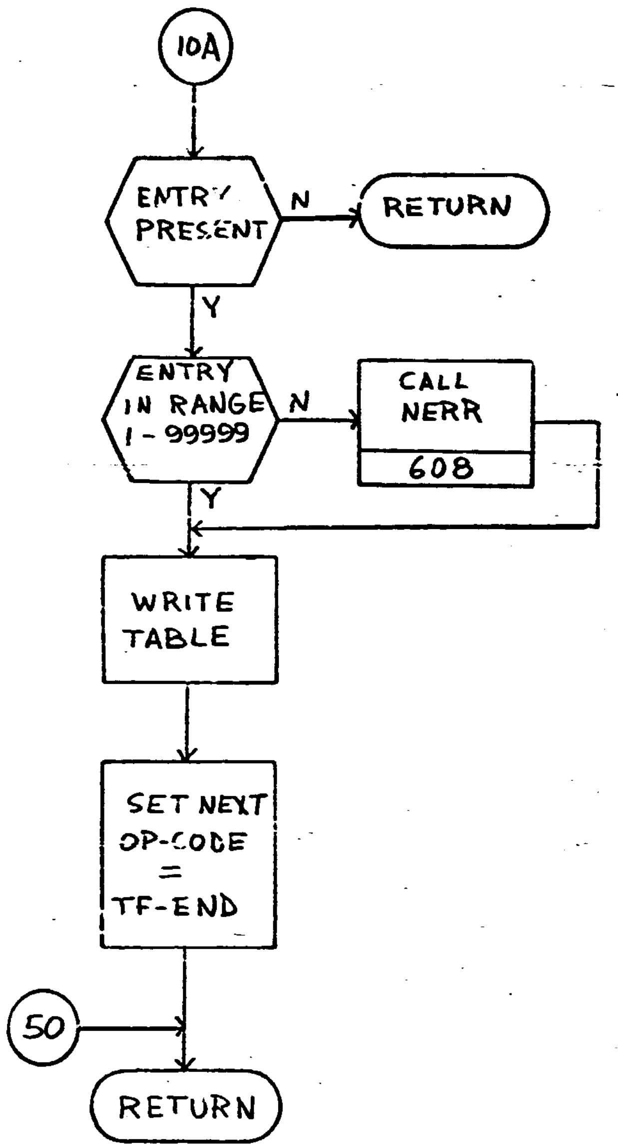

SUBROUTINE SX6
(TF-DEF CARD)
TF IN PROGRESS
Y
N
CALL NERR
601
RETURN
SET TF IN PROGRESS SWITCH
CALL GGSYM
FIND DEVICE NAME
FOUND
N
Y
CALL NERR
602
10
5


5
2ND OPERAND 0<X<14
N
Y
CALL NERR
604
10
SET NEXT OP-CODE = TABLE
10
SET ROW COUNTER TO ZERO
RETURN

SUBROUTINE SX7

This routine processes the TABLE statements. The row counter is incremented, and a test is made to see if there are more rows than files. The output area is then zeroed. The device number, row count, and number of files are stored in an appropriate table. The table entries are then converted using the CONV routine. The table is then written out to the simulator input tape.

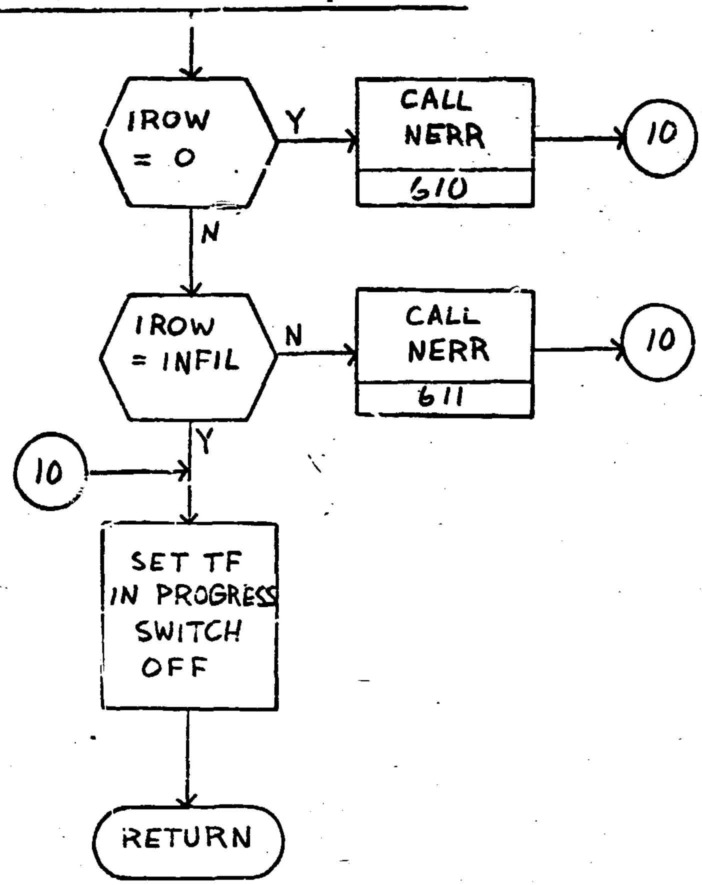

SUBROUTINE SX7
(TABLE)
ADD ONE TO ROW COUNTER
TO-ROW COUNTER > NBR OF FILES
Y
CALL NERR
606
50
N
ZERO OUTPUT AREA
STORE OPERANDS INTO TF-TABLE
SET POINTER TO 4
10A


10A
ENTRY PRESENT
N
RETURN
Y
ENTRY IN RANGE 1 - 99999
N
CALL NERR
608
Y
WRITE TABLE
SET NEXT OP-CODE = TF-END
50
RETURN

SUBROUTINE SX8

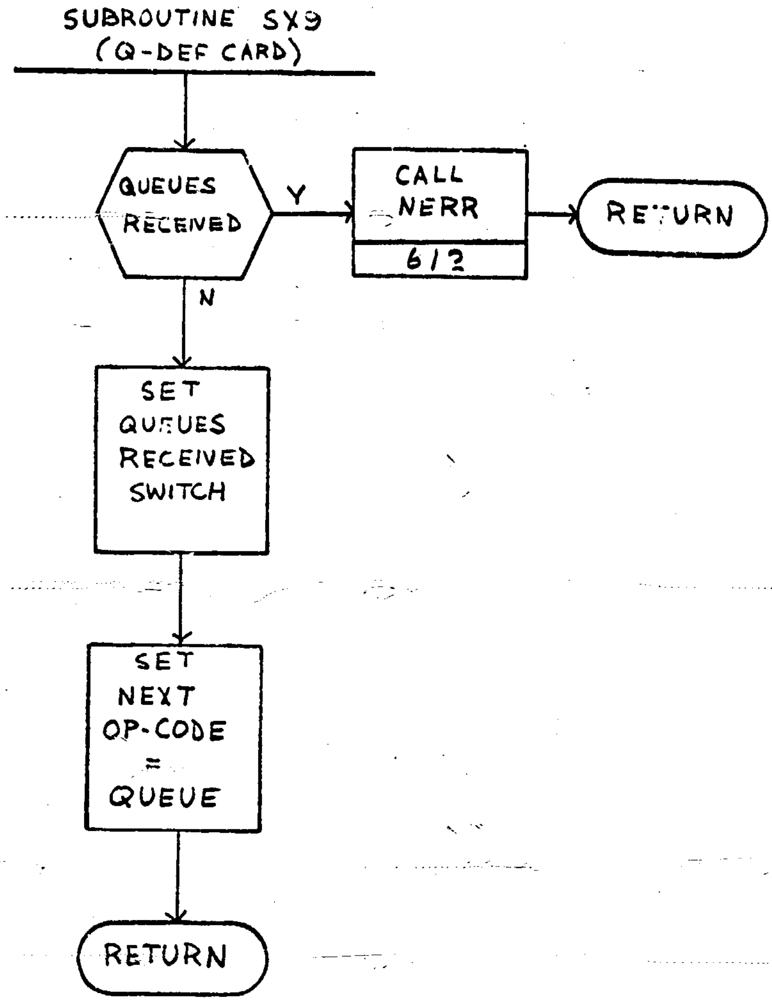
This routine processes the TF-END statement. If no table statements have been received, an error is written. A switch is set to indicate all table statements have now been received. A return is then made to the assembler first pass routine.

SUBROUTINE SX8
(TF-END CARD)



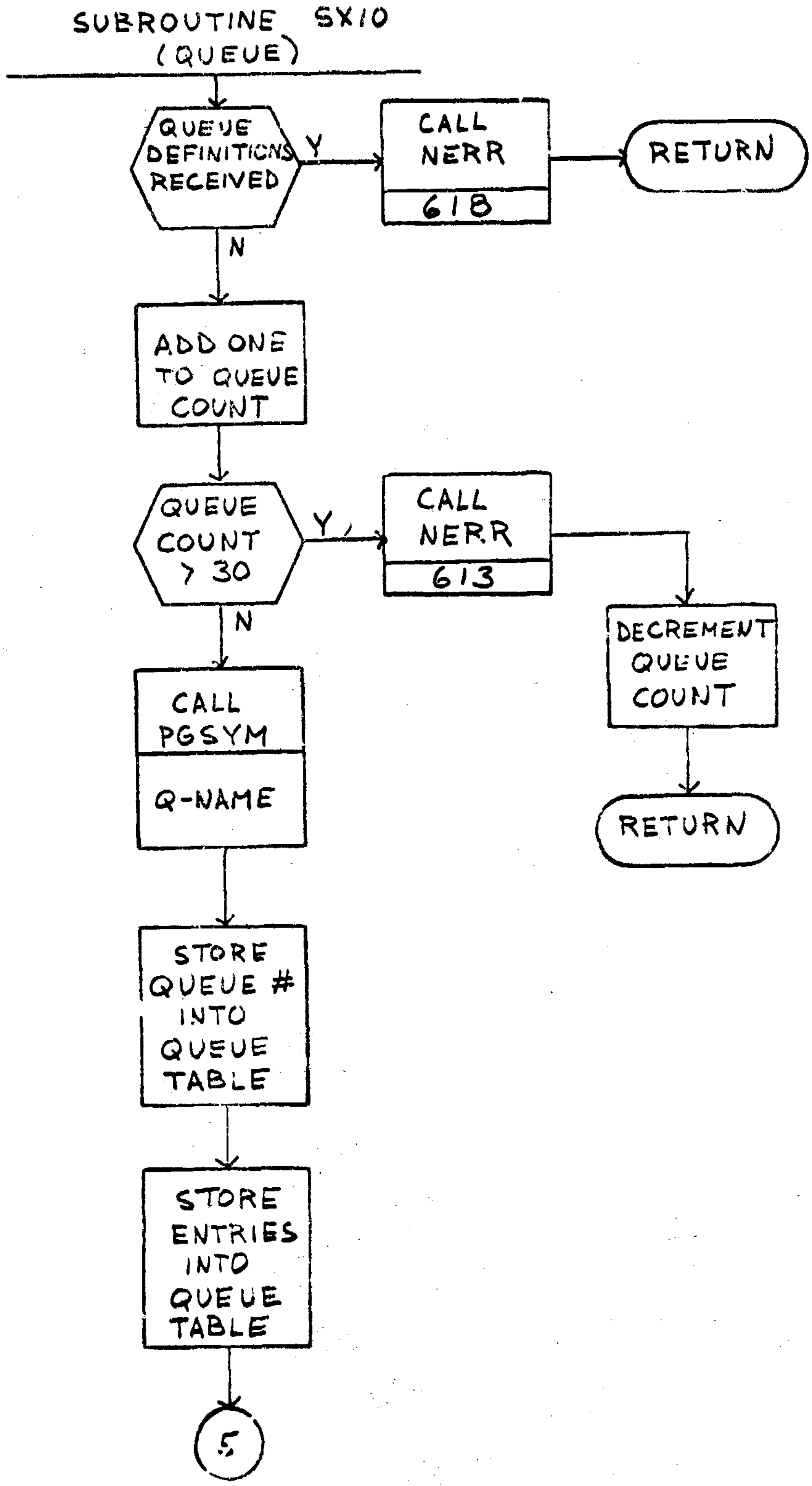
SUBROUTINE SX9

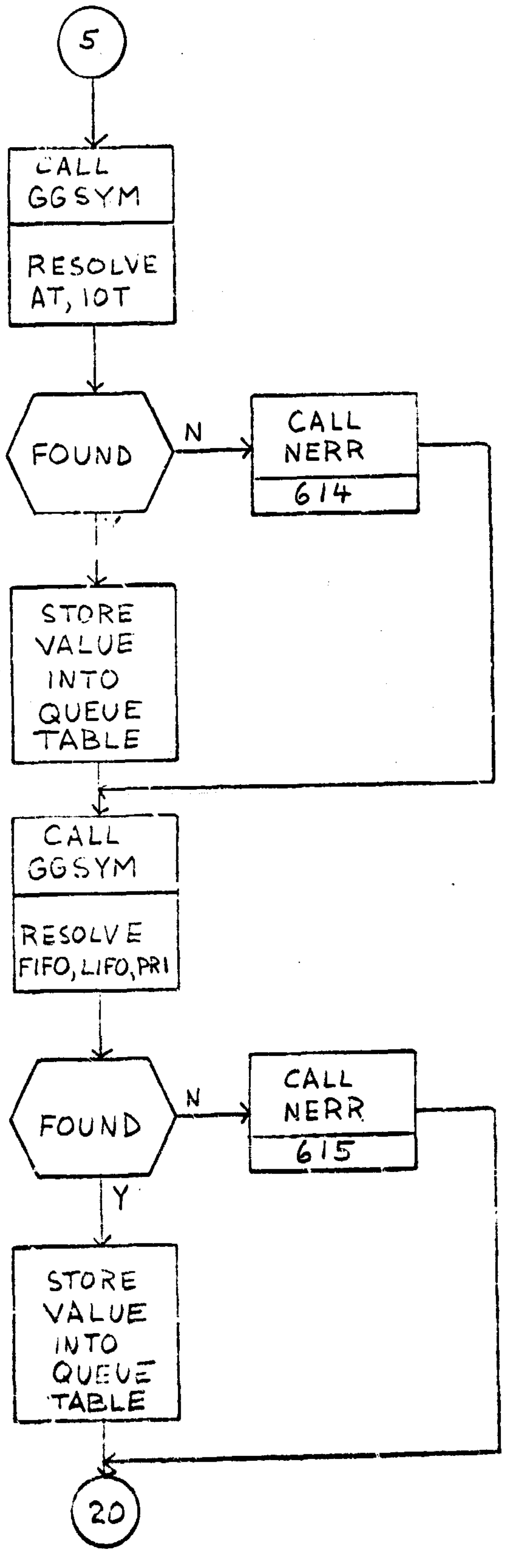
This routine is used to process the Q-DEF statement. It first tests to see if the queues have already been received. If they have, an error is written. If not, a switch is set to indicate that queues may now be received. The next anticipated op-code is set to be the QUEUE statement. A return is then made to the assembler first pass subroutine.

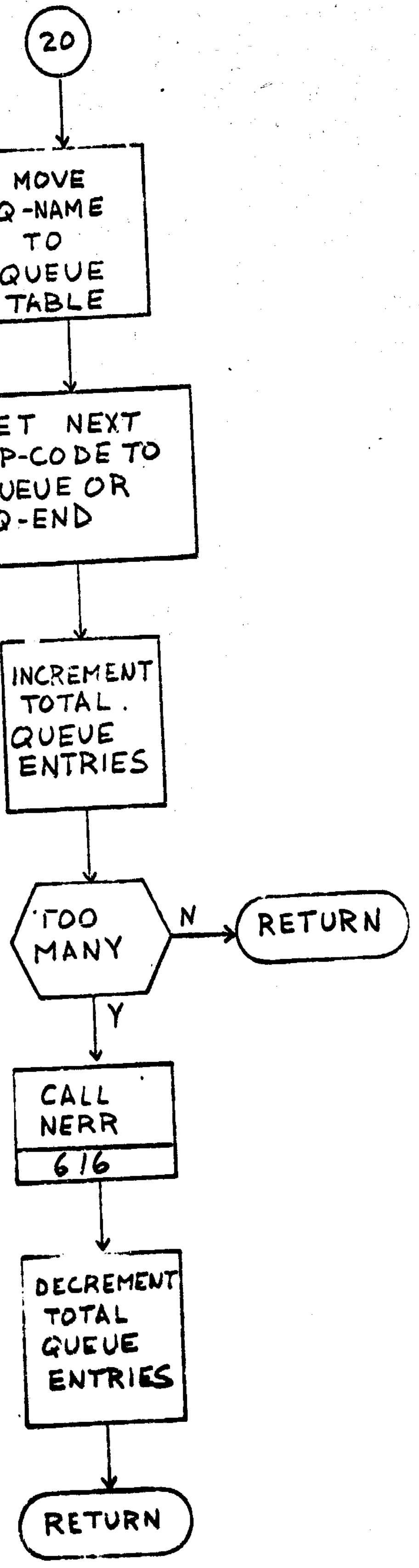

SUBROUTINE SX9
(Q-DEF CARD)
QUEUES RECEIVED
Y
N
CALL NERR
612
RETURN
SET QUEUES RECEIVED SWITCH
SET NEXT OP-CODE = QUEUE
RETURN

SUBROUTINE SX10

This routine processes the queue statement. It places a queue name into the global symbol table. After performing some housekeeping functions, a queue name is moved to the queue table. The next anticipated op-code is set to be either another QUEUE or a Q-END statement. A count is made of the total number of queue entries and a test is made to see if the maximum number has been exceeded.


SUBROUTINE SX10
(QUEUE)
QUEUE DEFINITIONS RECEIVED
Y
CALL NERR
618
RETURN
N
ADD ONE TO QUEUE COUNT
QUEUE COUNT > 30
Y
CALL NERR
613
DECREMENT QUEUE COUNT
RETURN
N
CALL PGSYM
Q-NAME
STORE QUEUE # INTO QUEUE TABLE
STORE ENTRIES INTO QUEUE TABLE
5

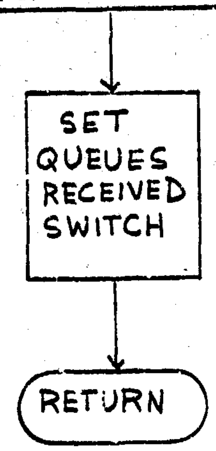

5
CALL
GGSYM
RESOLVE
AT, IOT
FOUND
N
CALL
NERR
614
STORE
VALUE
INTO
QUEUE
TABLE
CALL
GGSYM
RESOLVE
FIFO, LIFO, PRI
FOUND
N
CALL
NERR
615
Y
STORE
VALUE
INTO
QUEUE
TABLE
20


20
MOVE Q-NAME TO QUEUE TABLE
SET NEXT OP-CODE TO QUEUE OR Q-END
INCREMENT TOTAL QUEUE ENTRIES
TOO MANY
N
RETURN
Y
CALL NERR
616
DECREMENT TOTAL QUEUE ENTRIES
RETURN

SUBROUTINE SX11

The IQR switch is set to 2 indicating that all queues have been received. This routine is called when the Q-END statement is received.

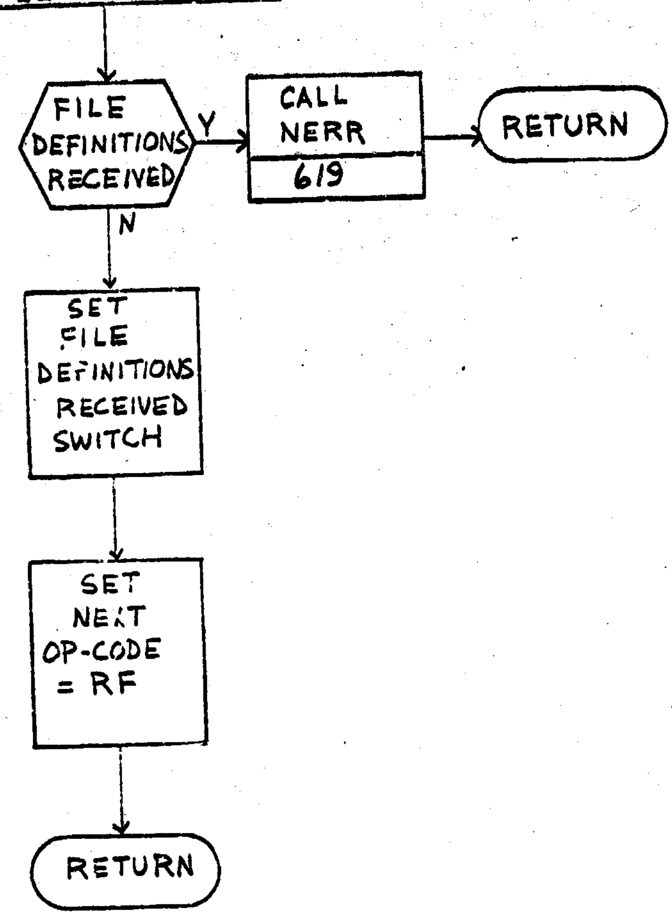
SUBROUTINE SX11
(Q-END)



SUBROUTINE SX12

This routine processes the FILE-DEF statement. A test is made to see if the file definitions have already been received. If they have, an error is declared. If not, a switch is set indicating that files are now to be received. The next op-code switch is set to accept RF statements.

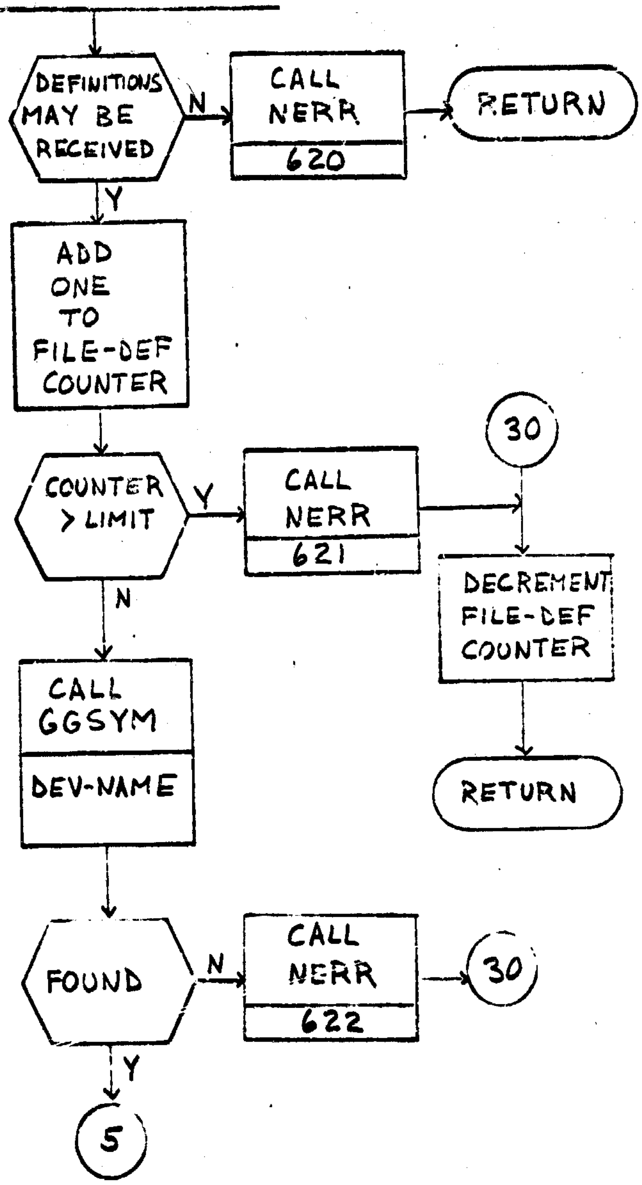
SUBROUTINE SX12
(FILE-DEF)

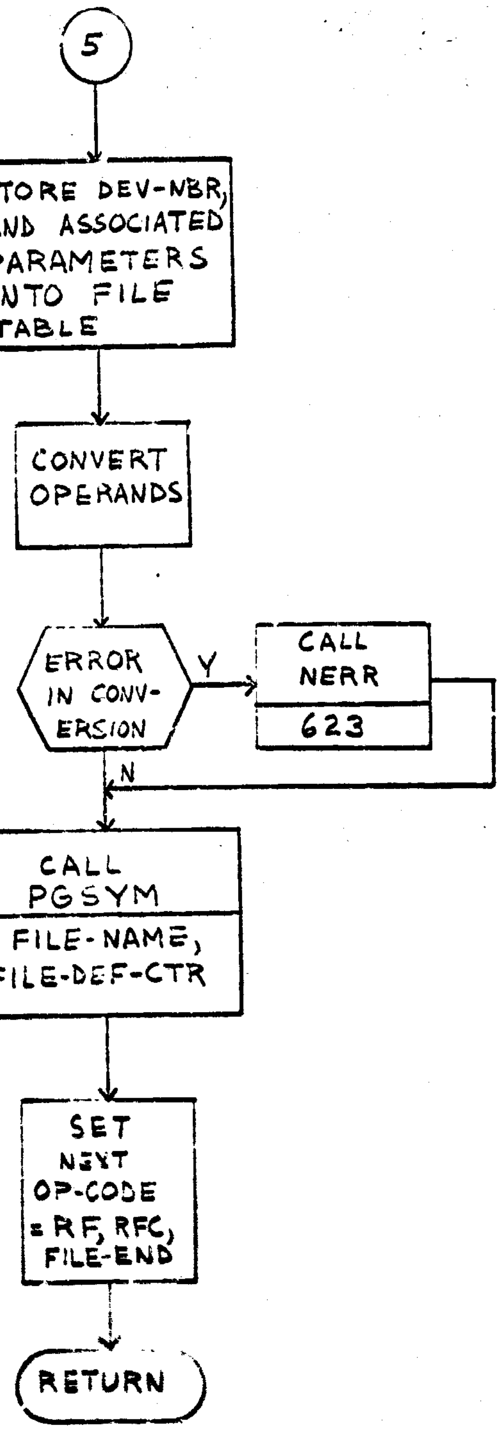


SUBROUTINE SX13

This routine processes RF statements. A test is made to see if file definitions are now in progress. A test is made to see if the maximum number of files is being exceeded. The device name is resolved, the file name and converted operands are stored in an appropriate table. The next op-code is set to be either another RF, RFC, or FILE-END statement.

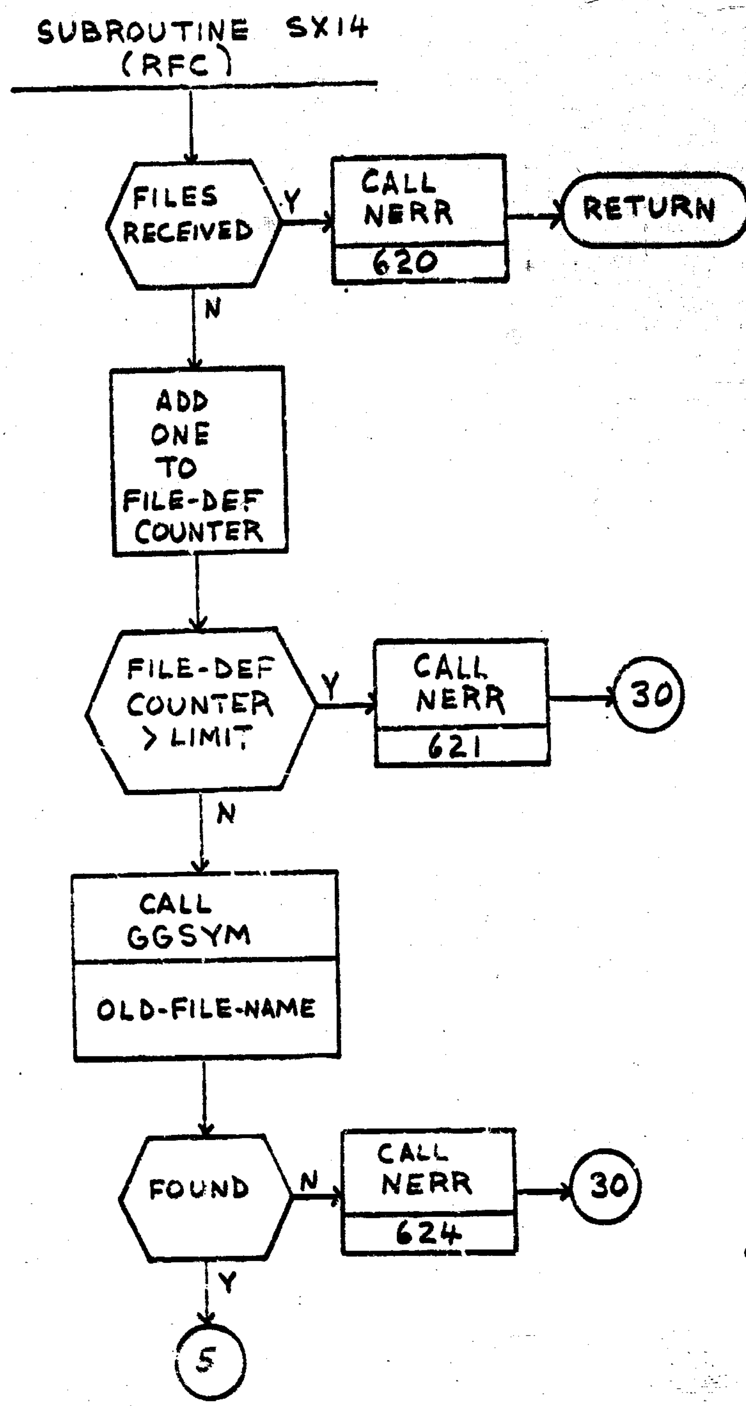
SUBROUTINE SX13
(RF)

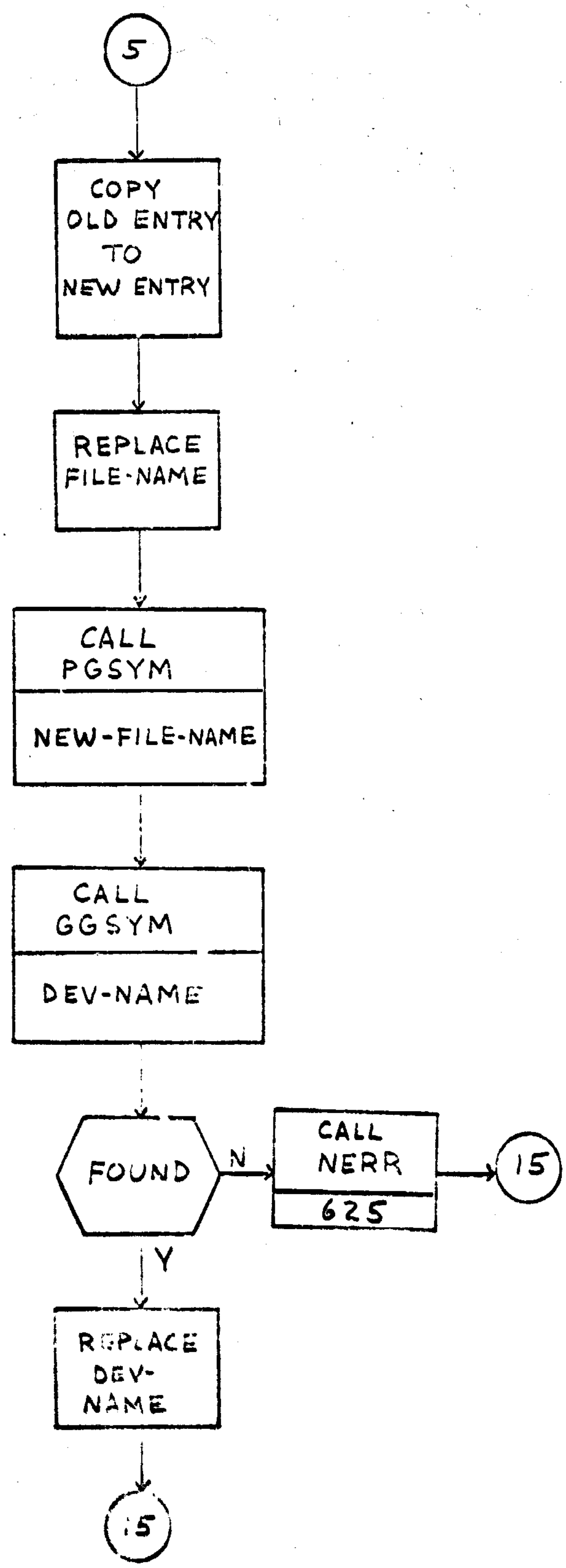


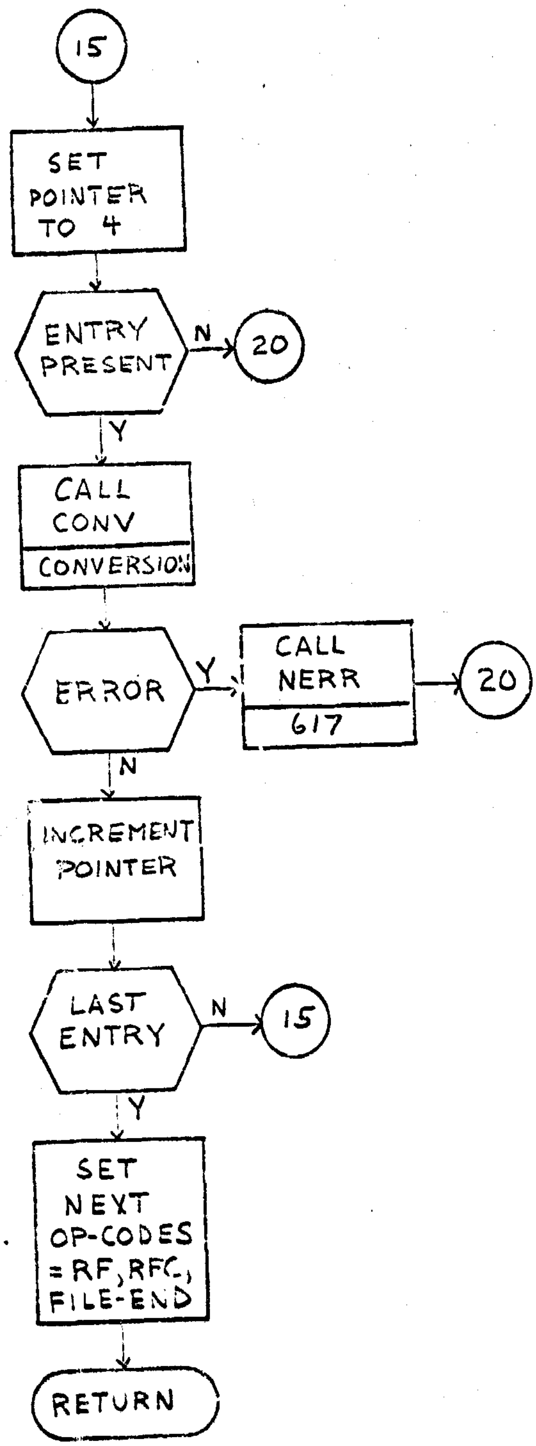

5
STORE DEV-NBR, AND ASSOCIATED PARAMETERS INTO FILE TABLE
CONVERT OPERANDS
ERROR IN CONVERSION
Y
N
CALL NERR
623
CALL PGSYM
FILE-NAME, FILE-DEF-CTR
SET NEXT OP-CODE = RF, RFC, FILE-END
RETURN

SUBROUTINE SX14

This routine processes the RFC statement. A test is first made to see if the file definition is in progress. A check is made to see that the maximum number of files has not been exceeded. A test is made to see if the old file name is valid. The new file name is put into the file table and the global symbol table. The device name is resolved into a device number and stored in the file table. The next expected op-codes are set to be RF, RFC, or file end.


SUBROUTINE SX14
(RFC)
FILES RECEIVED
Y
N
CALL NERR
620
RETURN
ADD ONE TO FILE-DEF COUNTER
FILE-DEF COUNTER > LIMIT
Y
N
CALL NERR
621
30
CALL GGSYM
OLD-FILE-NAME
FOUND
N
Y
CALL NERR
624
30
5

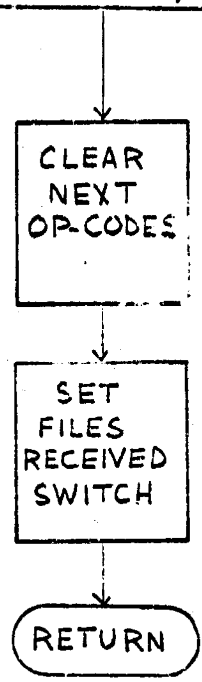

5
COPY OLD ENTRY TO NEW ENTRY
REPLACE FILE-NAME
CALL PGSYM
NEW-FILE-NAME
CALL GGSYM
DEV-NAME
FOUND
N
CALL NERR
625
15
Y
REPLACE DEV-NAME
15


15
SET POINTER TO 4
ENTRY PRESENT
N
20
Y
CALL CONV
CONVERSION
ERROR
Y
CALL NERR
617
20
N
INCREMENT POINTER
LAST ENTRY
N
15
Y
SET NEXT OP-CODES = RF, RFC, FILE-END
RETURN

SUBROUTINE SX15

This routine processes the file end statement. Its functions are to clear the next anticipated op-code area, and to set a switch indicating that file definitions have been received. A return is then made to the first pass subroutine.

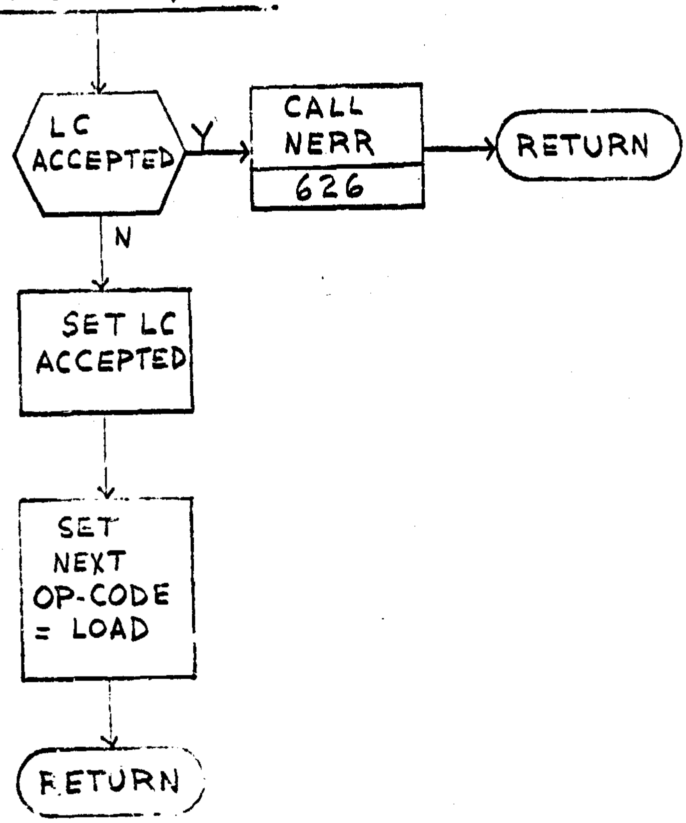
SUBROUTINE SX15
(FILE-END)



SUBROUTINE SX16

This routine processes the load class definition statement. A test is made to see if the load class information has already been received. If it has, an error is written. Otherwise, the load class switch is set to indicate that load class information may now be received. The next anticipated op-code is set to be the LOAD op-code. A return is made to the first pass subroutine.

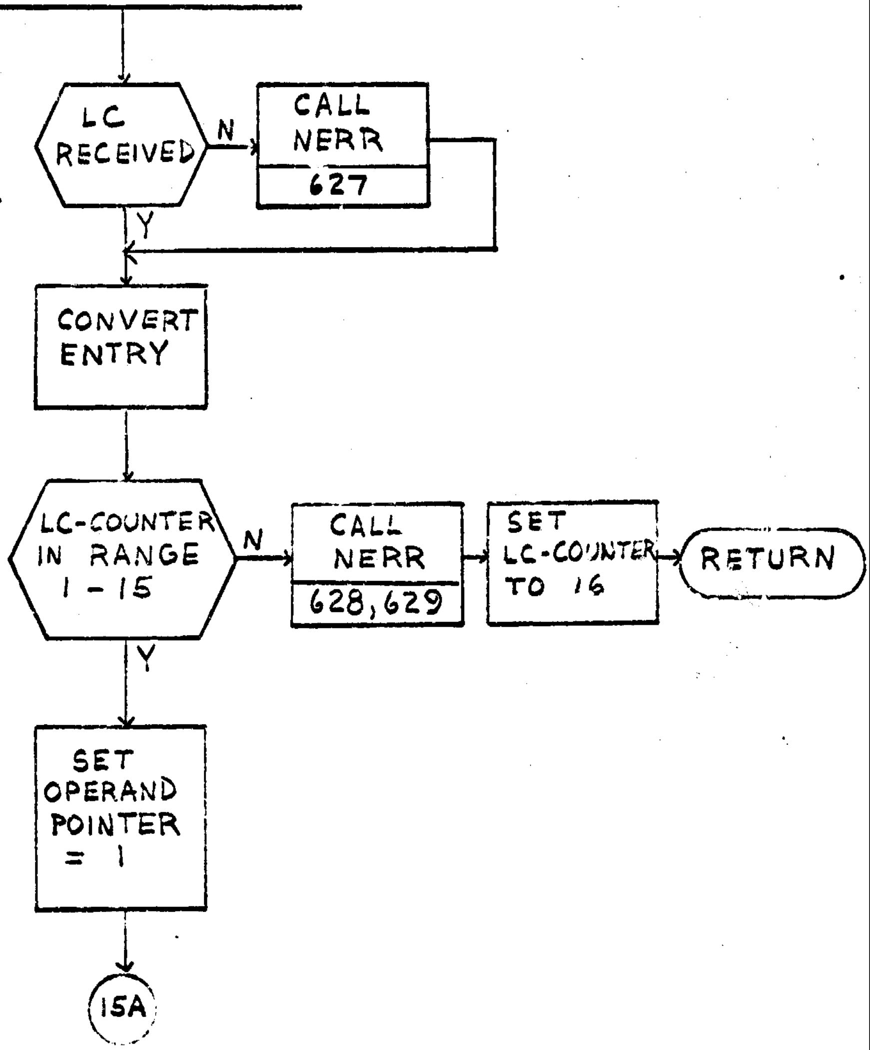
SUBROUTINE SX16
(LC-DEF)

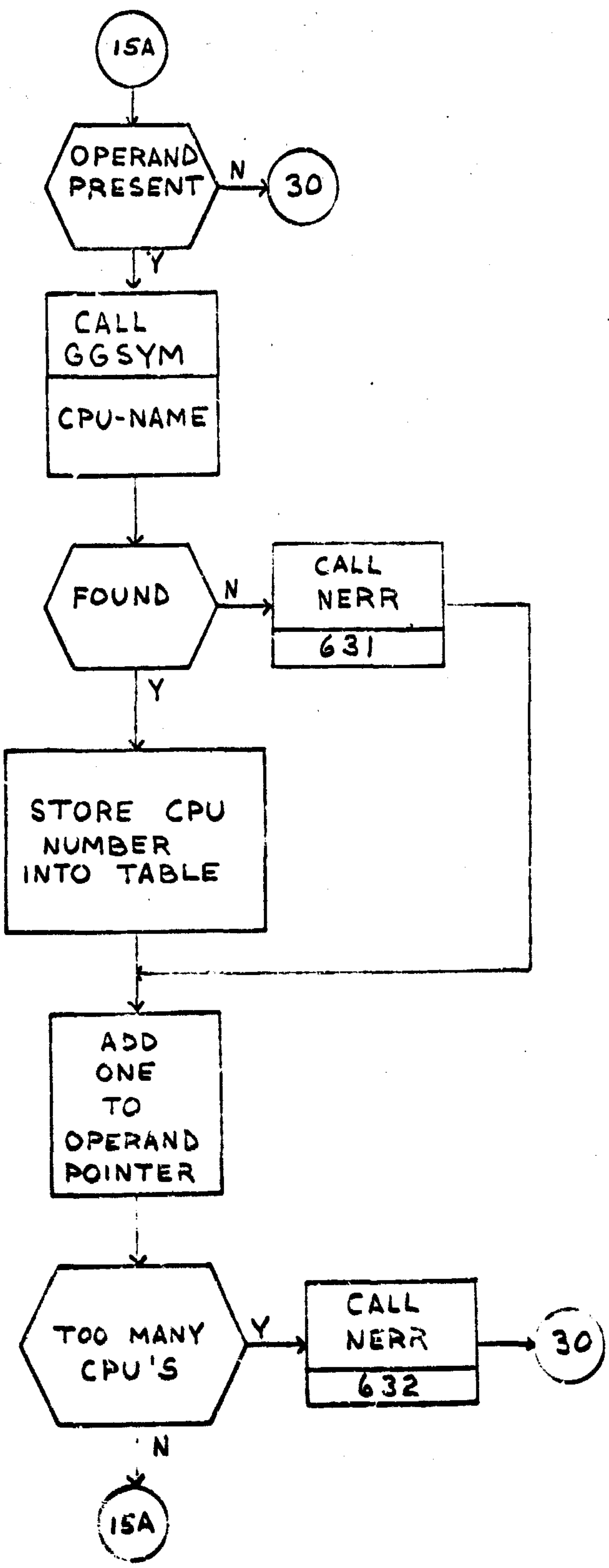


SUBROUTINE SX17

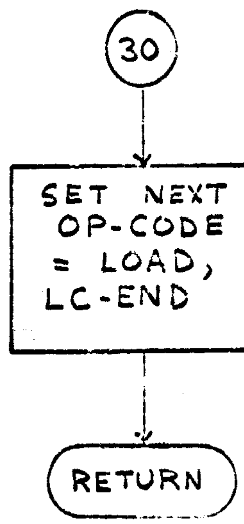
This routine processes the LOAD statement. A test is made to see if the load class information may now be received. The run class identification is then checked to see if it is within range, and if it has already been used. The CPU-IDs associated with this load class are then identified by using the global symbol table. Information is stored in an appropriate table. The next anticipated op-code is set to be either LOAD or LC-END.

SUBROUTINE SX17
(LOAD)



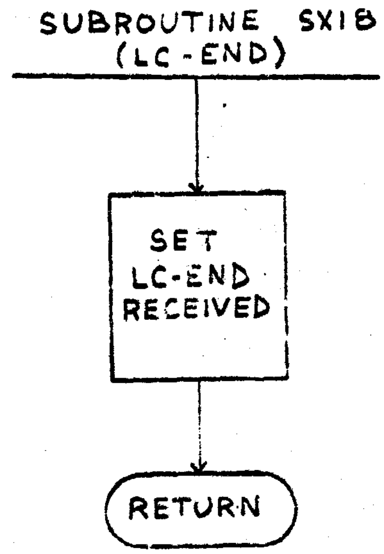


ISA
OPERAND PRESENT
N
30
Y
CALL GGSYM
CPU-NAME
FOUND
N
CALL NERR
631
Y
STORE CPU NUMBER INTO TABLE
ADD ONE TO OPERAND POINTER
TOO MANY CPU'S
Y
CALL NERR
632
30
N
ISA


30
SET NEXT OP-CODE = LOAD, LC-END
RETURN

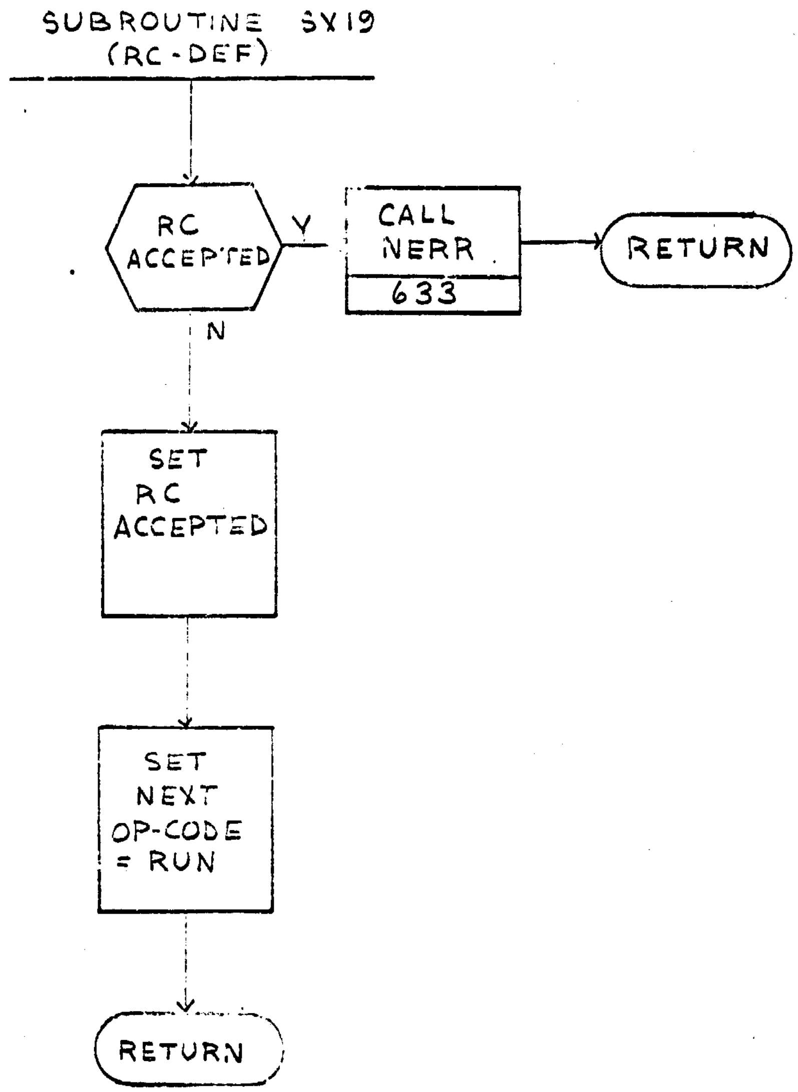
SUBROUTINE SX18

This routine processes the LC-END statement. This routine indicates that the end of the load class data has been received. This is indicated by setting a switch. A return is then made to the first pass subroutine.


SUBROUTINE SX1B
(LC-END)
SET
LC-END
RECEIVED
RETURN

SUBROUTINE SX19

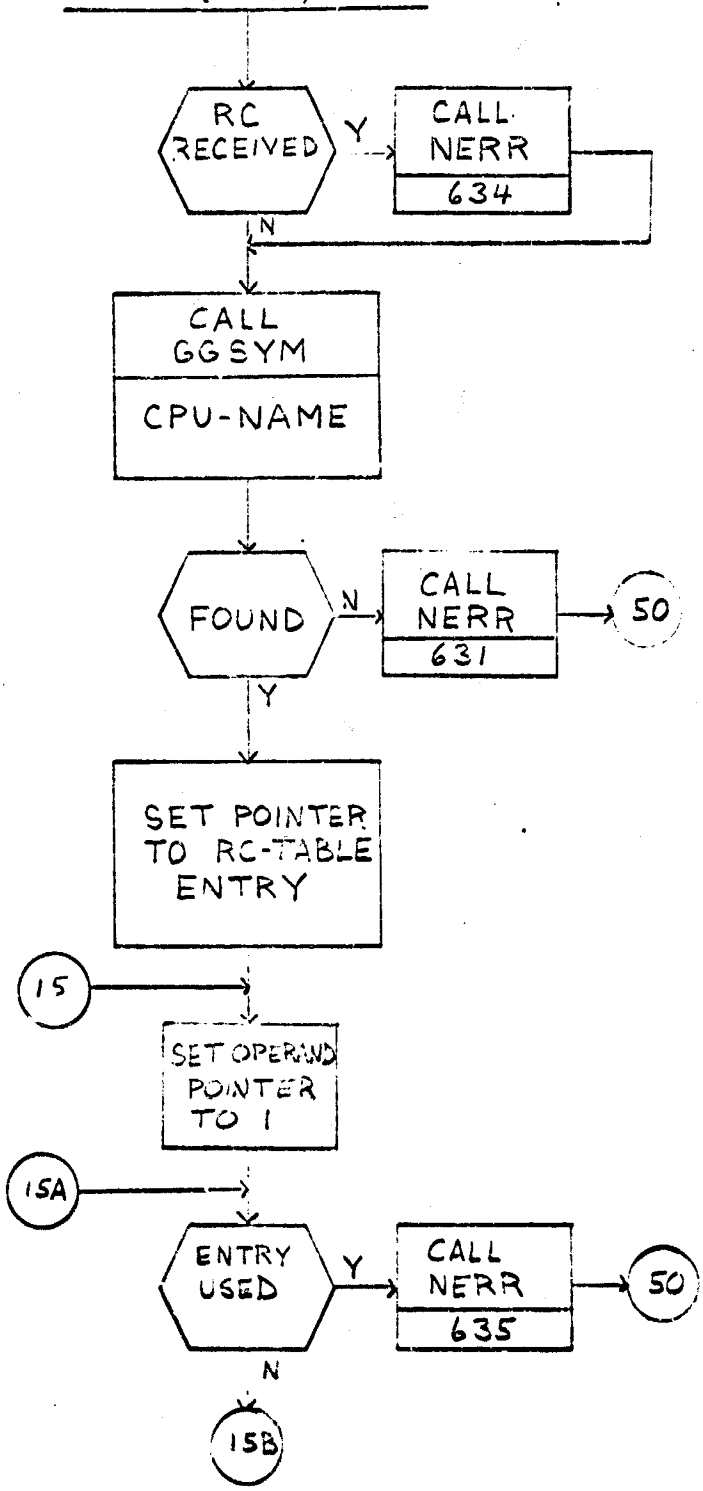
This routine processes the run class definition card. A test is first made to see whether the run class information has already been accepted. If it has, then an error is written. Otherwise, a switch is set to indicate that run class definitions may now be received. The next anticipated op-code is set to RUN. A return is made to the first pass subroutine.

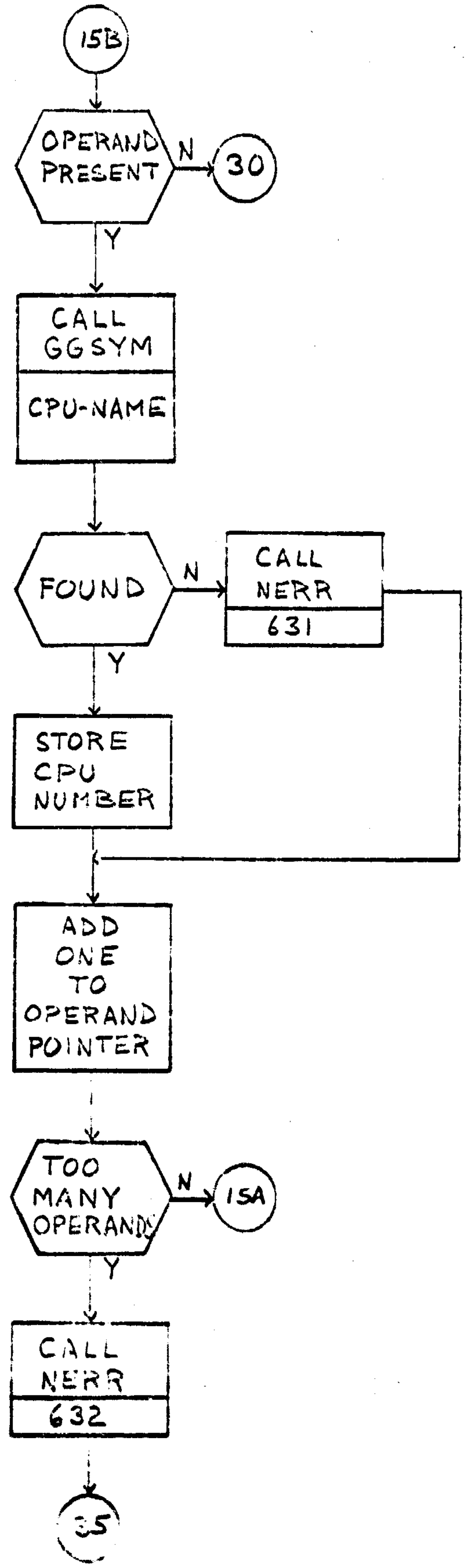

SUBROUTINE SX19
(RC-DEF)
RC ACCEPTED
Y
N
CALL NERR
633
RETURN
SET RC ACCEPTED
SET NEXT OP-CODE = RUN
RETURN

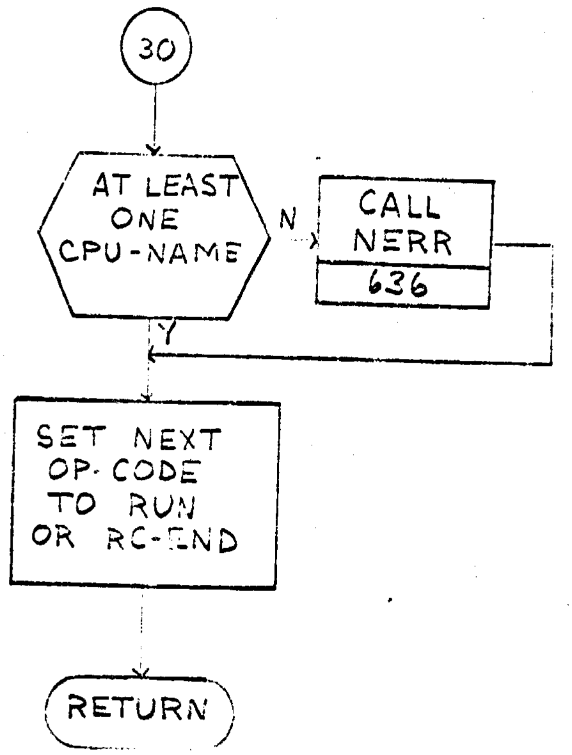
SUBROUTINE SX20

This routine processes the RUN statement. A test is first made to see whether run definitions may now be accepted. If not, an error is written. The CPU names that are associated with this run card are identified by use of the global symbol table. The identity of the CPUs is stored in an appropriate table. The next op-code is set to be either RUN or RC-END.

SUBROUTINE SX20
(RUN)

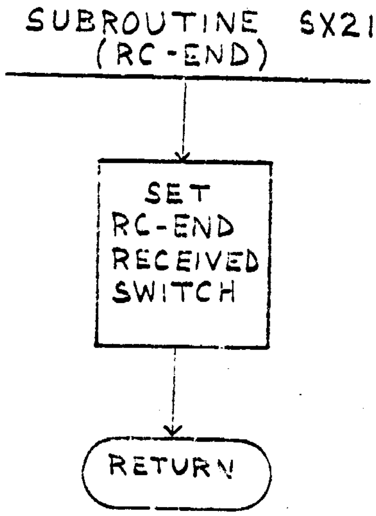



15B
OPERAND PRESENT
N
30
Y
CALL GGSYM
CPU-NAME
FOUND
N
CALL NERR
631
Y
STORE CPU NUMBER
ADD ONE TO OPERAND POINTER
TOO MANY OPERANDS
N
15A
Y
CALL NERR
632
35


30
AT LEAST ONE CPU-NAME
N
Y
CALL NERR
636
SET NEXT OP-CODE TO RUN OR RC-END
RETURN

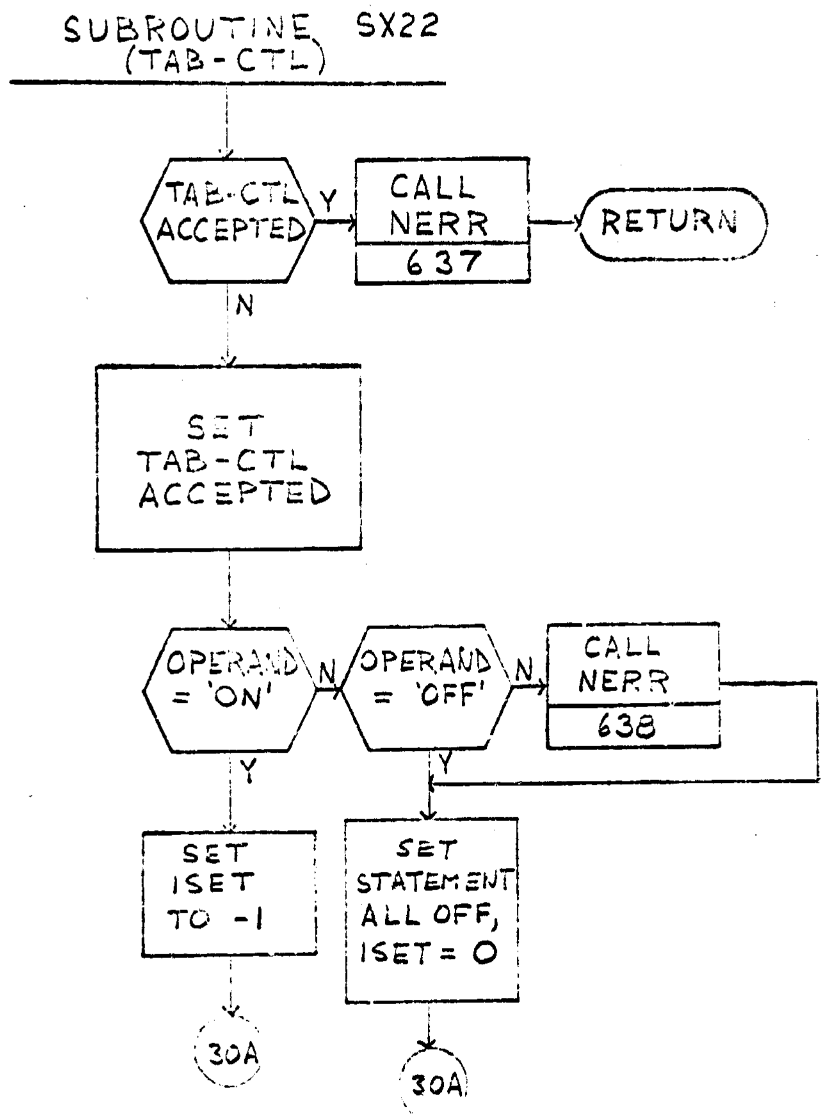
SUBROUTINE SX21

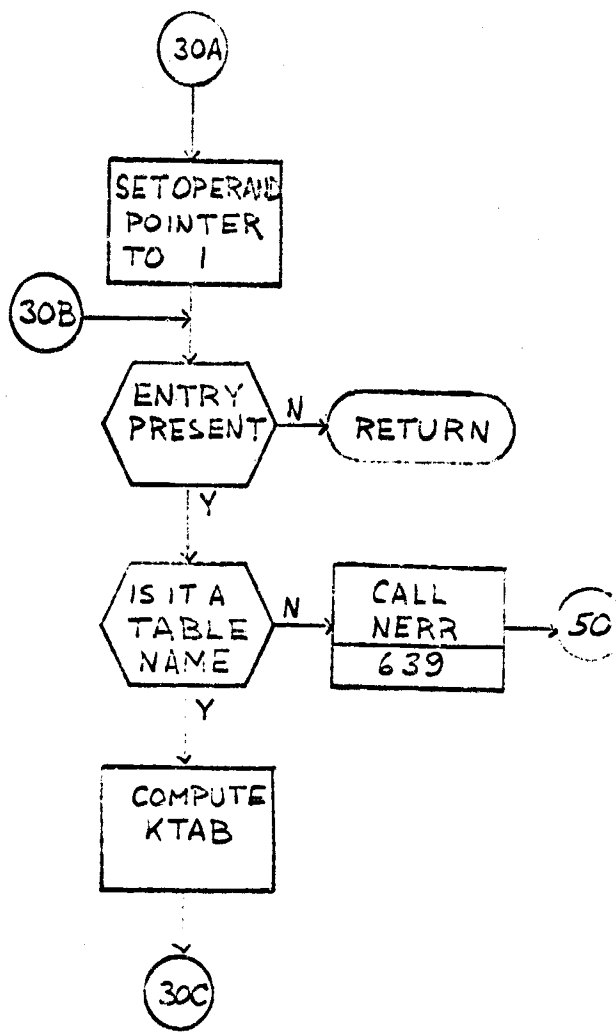
This routine processes the RC-END statement. This statement denotes the end of the RUN CLASS definition information. This condition is indicated by setting a switch. A return is then made to the first pass subroutine.

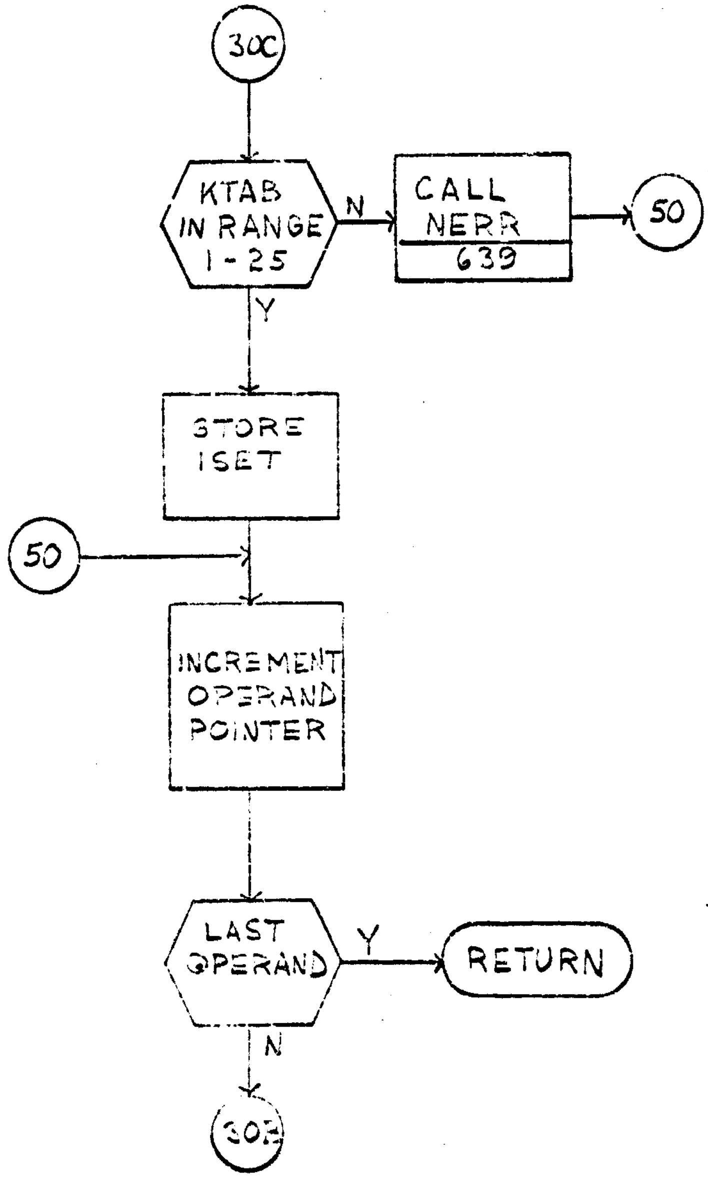

SUBROUTINE SX21
(RC-END)
SET
RC-END
RECEIVED
SWITCH
RETURN

SUBROUTINE SX22

This routine processes the table control statement. A check is first made to see whether or not a table control statement has already been accepted. If it has, an error is written. If not, a switch is set to indicate that the table control statement has been accepted. A test is made to see whether or not a normal condition is ON or OFF. Subsequent operands, denoting tables, are identified and their print status is set. A return is made to the first pass subroutine.

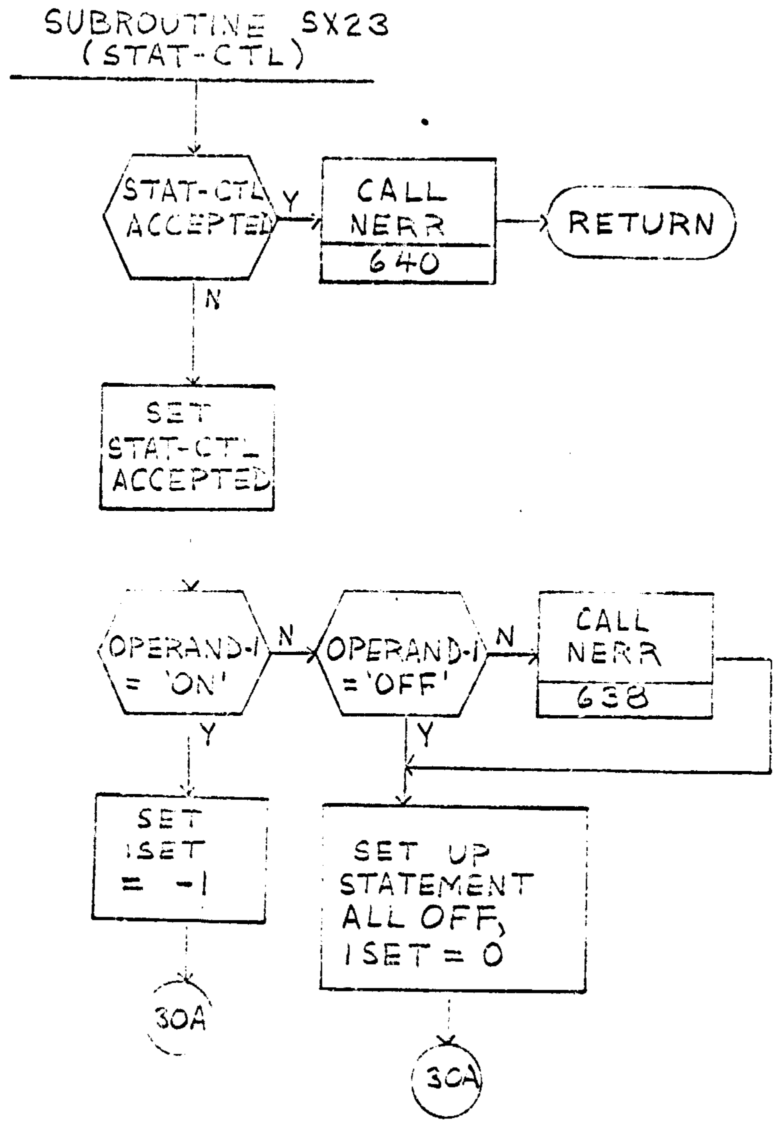

SUBROUTINE SX22
(TAB-CTL)
TAB-CTL ACCEPTED
Y
CALL NERR
637
RETURN
N
SET TAB-CTL ACCEPTED
OPERAND = 'ON'
N
OPERAND = 'OFF'
N
CALL NERR
638
Y
Y
SET ISET TO -1
SET STATEMENT ALL OFF, ISET = 0
30A
30A

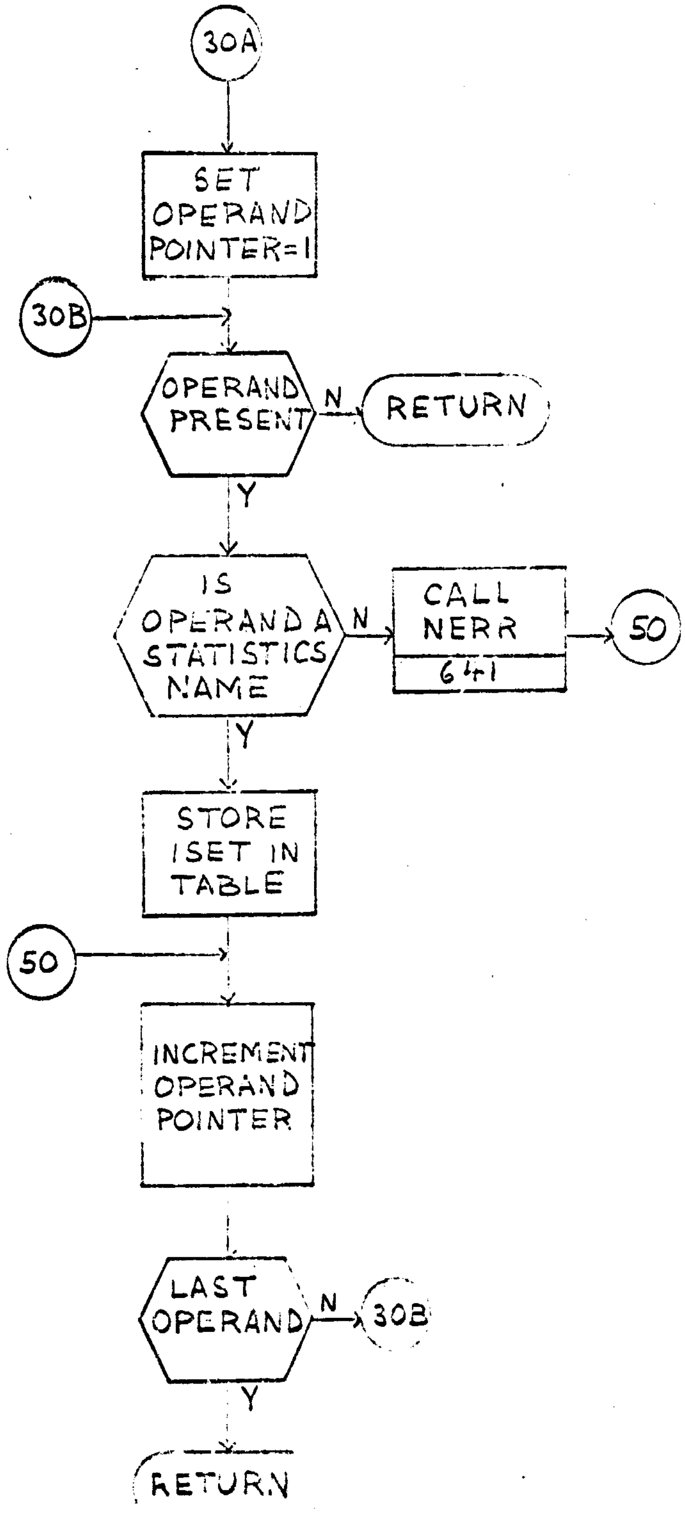

30A
SET OPERAND POINTER TO 1
30B
ENTRY PRESENT
N
RETURN
Y
IS IT A TABLE NAME
N
CALL NERR
639
50
Y
COMPUTE KTAB
30C


30C
KTAB IN RANGE 1-25
N
CALL NERR
639
50
Y
STORE ISET
50
INCREMENT OPERAND POINTER
LAST OPERAND
Y
RETURN
N
30B

SUBROUTINE SX23

This routine processes the statistics control statement. A test is first made to see whether or not the statistics control statement has already been received. If it has, an error condition is raised. If not, a switch is set to indicate that the statistics control statement has been received. A test is then made to see whether the normal condition for printing out a statistics table is ON or OFF. Subsequent operands, denoting statistics tables to be printed, are identified and their print status is set. A return is then made to the first pass subroutine.

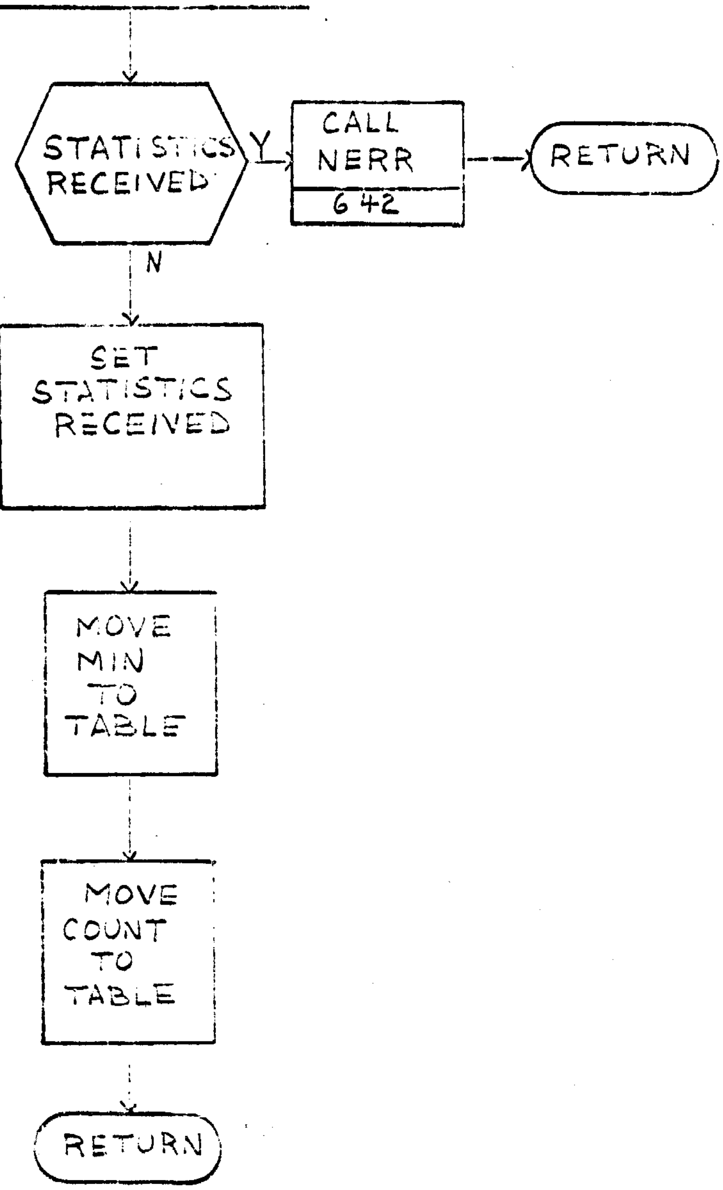

SUBROUTINE SX23
(STAT-CTL)
STAT-CTL ACCEPTED
Y
CALL NERR
640
RETURN
N
SET STAT-CTL ACCEPTED
OPERAND-1 = 'ON'
N
OPERAND-1 = 'OFF'
N
CALL NERR
638
Y
Y
SET ISET = -1
SET UP STATEMENT ALL OFF, ISET = 0
30A
30A


30A
SET OPERAND POINTER=1
30B
OPERAND PRESENT
N
RETURN
Y
IS OPERAND A STATISTICS NAME
N
CALL NERR
641
50
Y
STORE ISET IN TABLE
50
INCREMENT OPERAND POINTER
LAST OPERAND
N
30B
Y
RETURN

SUBROUTINE SX24

This routine processes the STATISTICS statement. This routine obtains the first and second operands denoting the statistics interval and the number of such intervals to be used during this simulation, and stores them in appropriate fields. A return is then made to the calling subroutine.

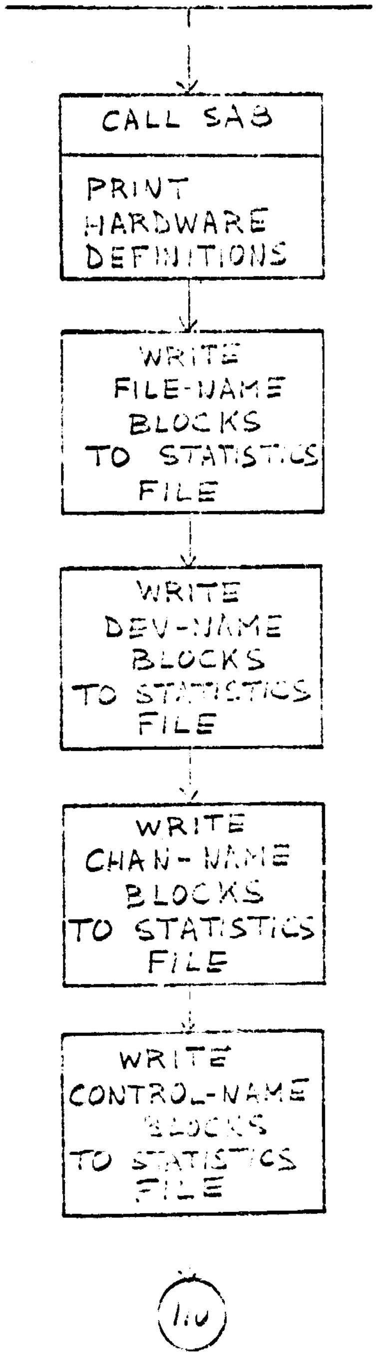
SUBROUTINE SX24
(STATISTICS)

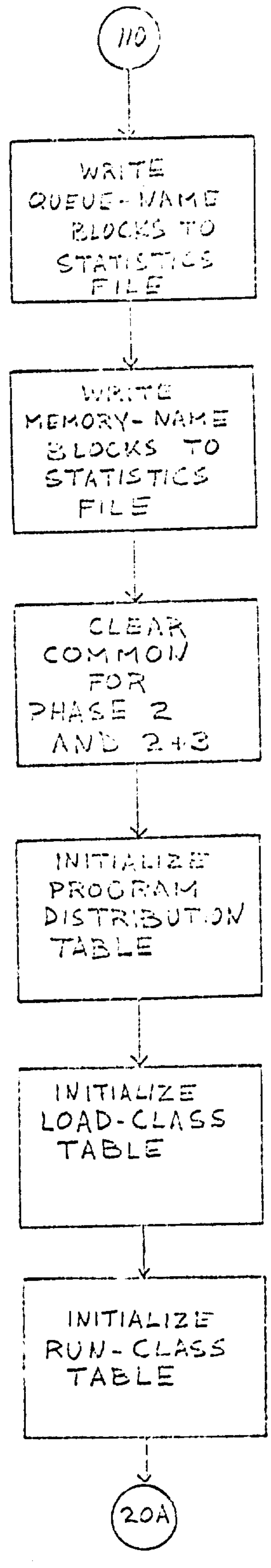


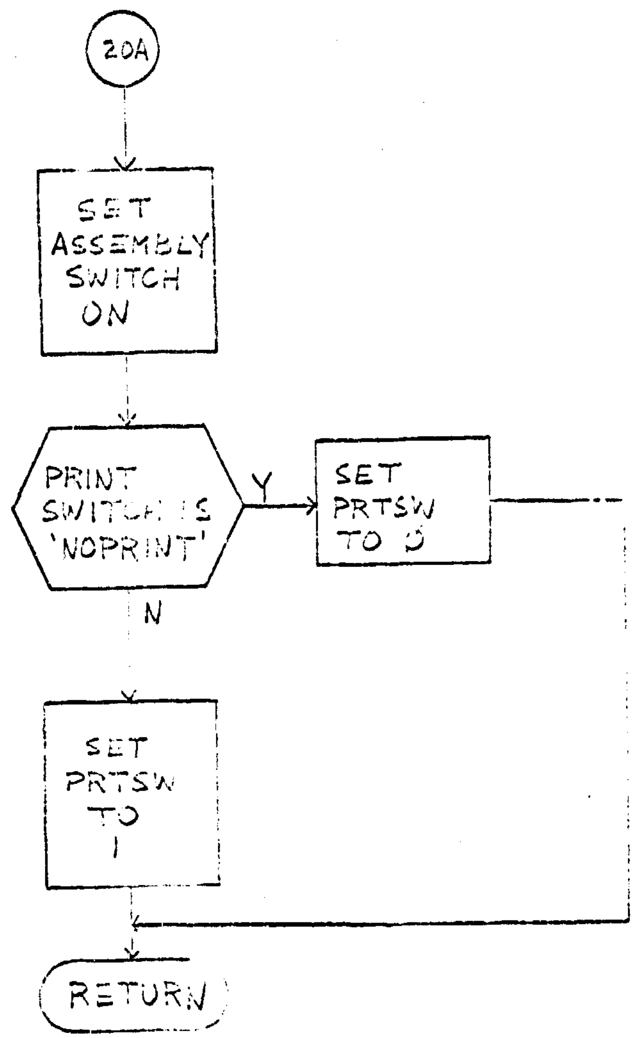
SUBROUTINE SX25

This routine processes the ASSEMBLY statement. It first calls SA8 which is used to print the hardware defintions. It then writes several blocks of data to the statistics file. These blocks contain file names, device names, channel names, control name blocks, queue name blocks, and memory name blocks. Common is then cleared. The program distribution table is then initialized. The load class and run class tables are also initialized. The assembly switch is set OFF to permit processing of worker program routines. A test is then made to see whether the operand in the assembly statement was NOPRINT, and the appropriate print switch setting is made. A return is then made to the first pass subroutine.

SUBROUTINE SX25
(ASSEMBLY)

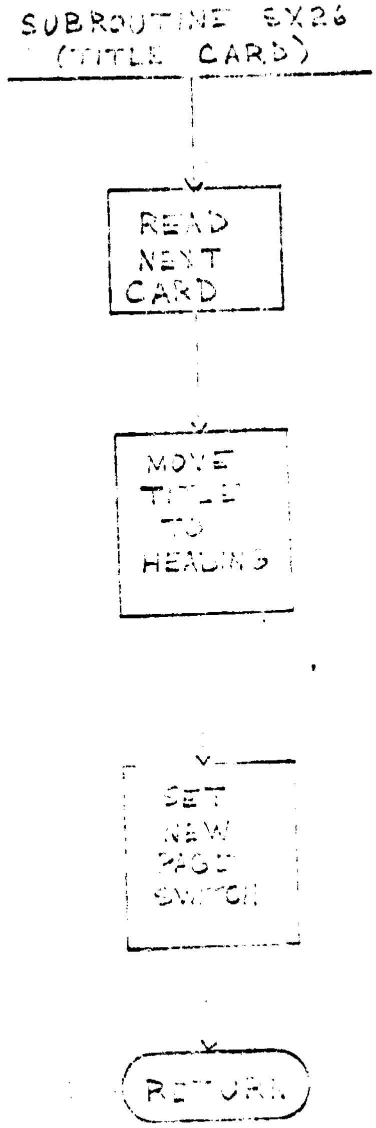



110
WRITE QUEUE-NAME BLOCKS TO STATISTICS FILE
WRITE MEMORY-NAME BLOCKS TO STATISTICS FILE
CLEAR COMMON FOR PHASE 2 AND 2+3
INITIALIZE PROGRAM DISTRIBUTION TABLE
INITIALIZE LOAD-CLASS TABLE
INITIALIZE RUN-CLASS TABLE
20A


20A
SET ASSEMBLY SWITCH ON
PRINT SWITCH IS 'NOPRINT'
Y
SET PRTSW TO 0
N
SET PRTSW TO 1
RETURN

SUBROUTINE SX26

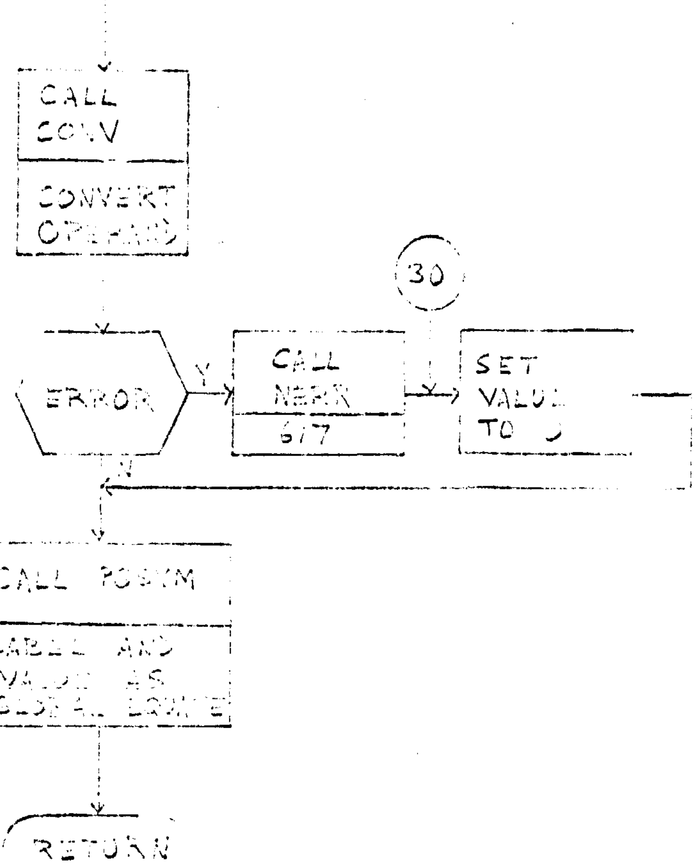
This routine processes the TITLE statement. The next card is read from the input stream and its contents stored in the page title area of memory. The printer is then advanced to the top of a new page and the title printed out there. A return is then made to the first pass subroutine.


SUBROUTINE EX26
(TITLE CARD)
READ NEXT CARD
MOVE TITLE TO HEADING
SET NEW PAGE SWITCH
RETURN

SUBROUTINE SX27

This routine processes the GEQU statement. This statement is used to define global equates. The first operand is converted to internal binary form, and the statement label and this value are stored in the global symbol table as a global equate. A return is then made to the calling subroutine.

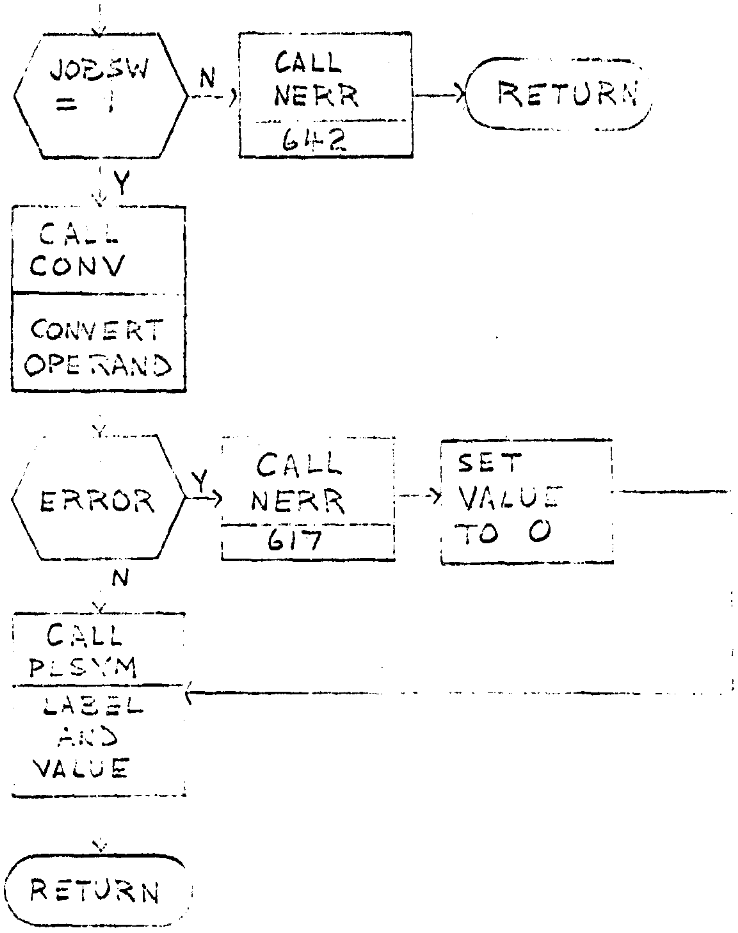
SUBROUTINE SX27
(GENU CARD)



SUBROUTINE SX28

This routine processes the LEQU statement. A test is first made to see whether a job statement has preceded this LEQU statement. If not, an error is written. Otherwise, the first operand is converted to internal binary form, and this value and the statement lable are stored in the local symbol table as a local equate. A return is then made to the first pass subroutine.

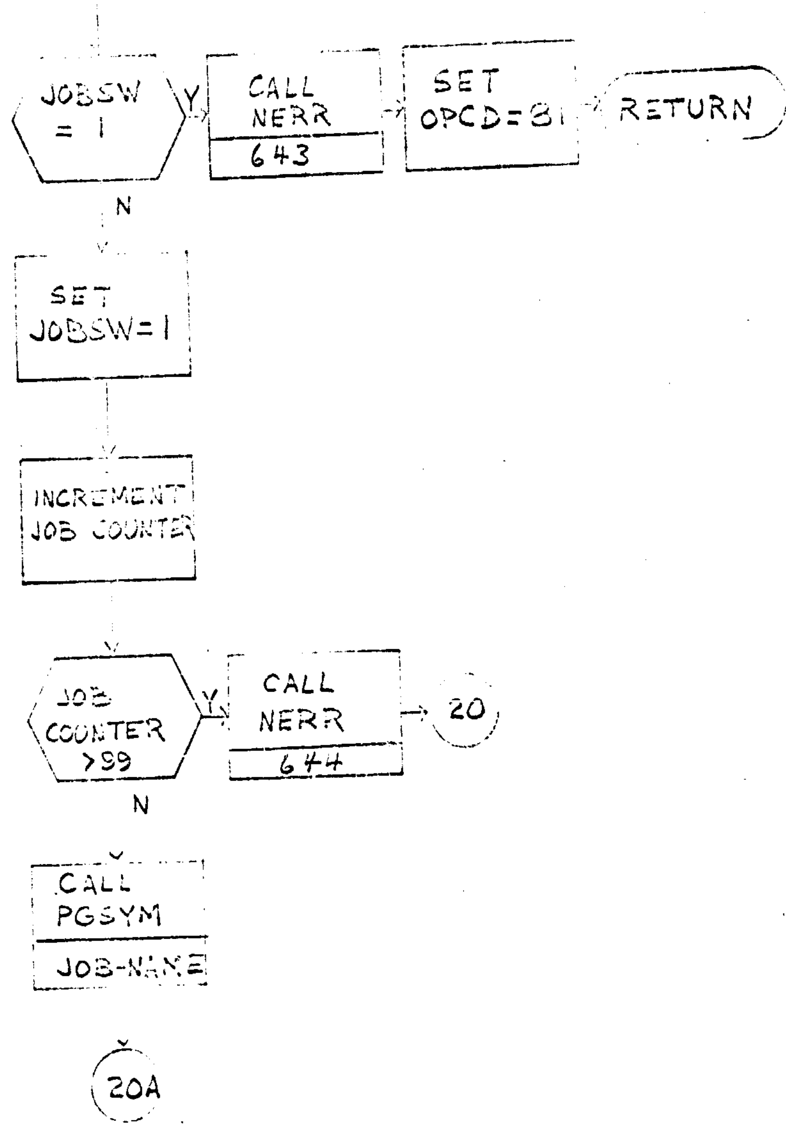
SUBROUTINE SX28
(LEQU CARD)

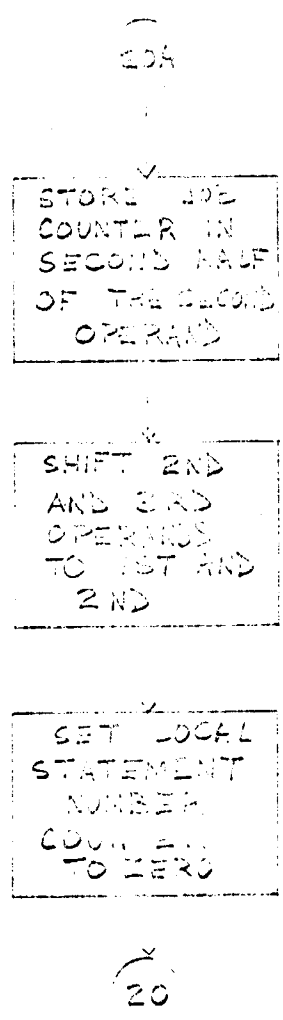


SUBROUTINE SX29

This routine processes the JOB statement. A test is made to see whether this statement has occurred in the middle of another job. If it has, an error is written. If not, a switch is set to indicate that a job has been begun. The number of jobs is then incremented by one. A test is then made to see whether the number of jobs received has exceeded the maximum. The job name is stored in the global symbol table. Some internal housekeeping is then performed. The local job statement number is then set to zero. A return is then made to the first pass subroutine.

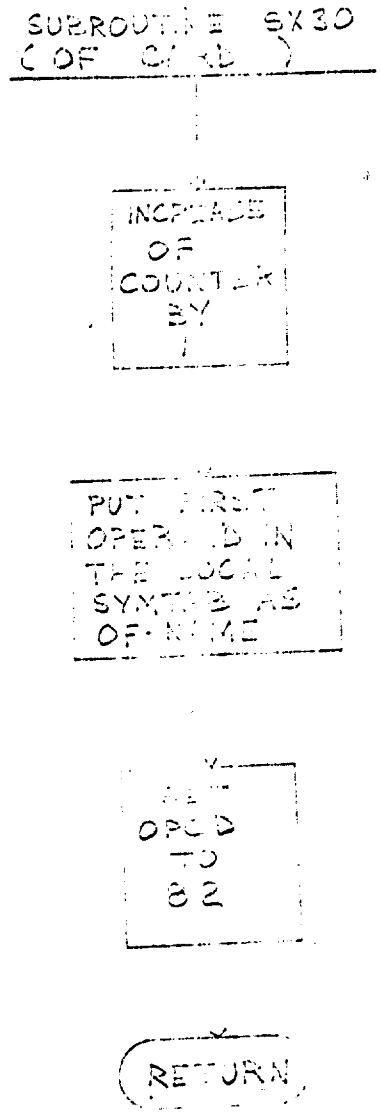
SUBROUTINE SX29
(JOB CARD)




20A
STORE THE COUNTER IN SECOND HALF OF THE SECOND OPERAND
SHIFT 2ND AND 3RD OPERANDS TO 1ST AND 2ND
SET LOCAL STATEMENT NUMBER COUNTER TO ZERO
20

SUBROUTINE SX30

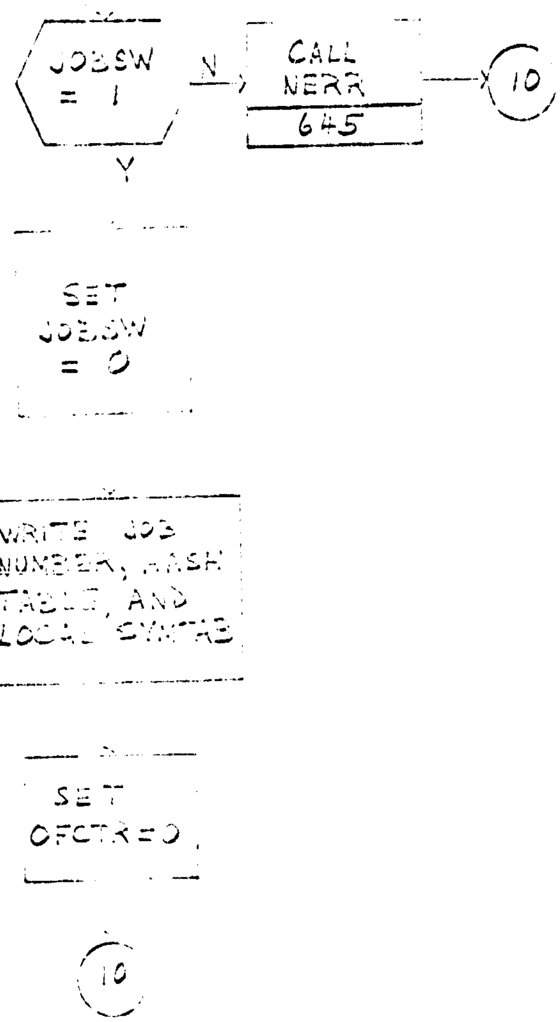
This routine processes the OF statement. The number of OF statements received is incremented by one. The OF name and its numeric value are then stored in the local symbol table. The op-code switch is set to 81 denoting an OF statement has been received. A return is then made to the first pass subroutine.

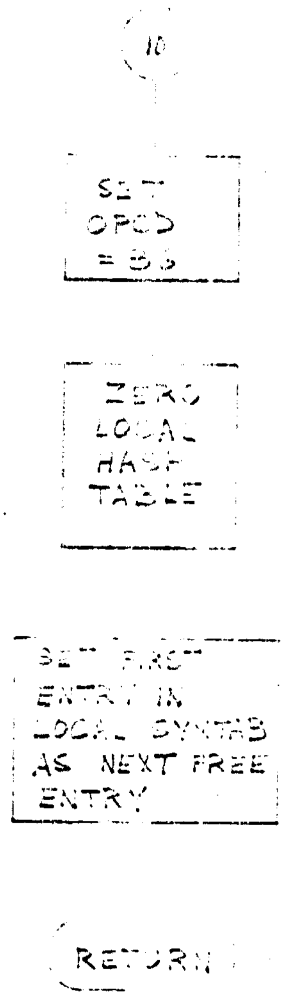

SUBROUTINE SX30
(OF CARD)
INCREASE OF COUNTER BY 1
PUT FIRST OPERAND IN THE LOCAL SYMTAB AS OF-NAME
[illegible] OPCD TO 82
RETURN

SUBROUTINE SX31

This routine processes the END statement. A check is first made to see whether a job is in progress. If not, an error message is written. The job switch is then set to zero, denoting that no job is in progress. The job identification number, the local hash table, and the local symbol table are then written out onto file number 8. The ordinal file counter is reset to zero. The op-code switch is then set to 86, denoting the fact that an END statement has been received. A return is then made to the first pass subroutine.

SUBROUTINE SX31
(END CARD)

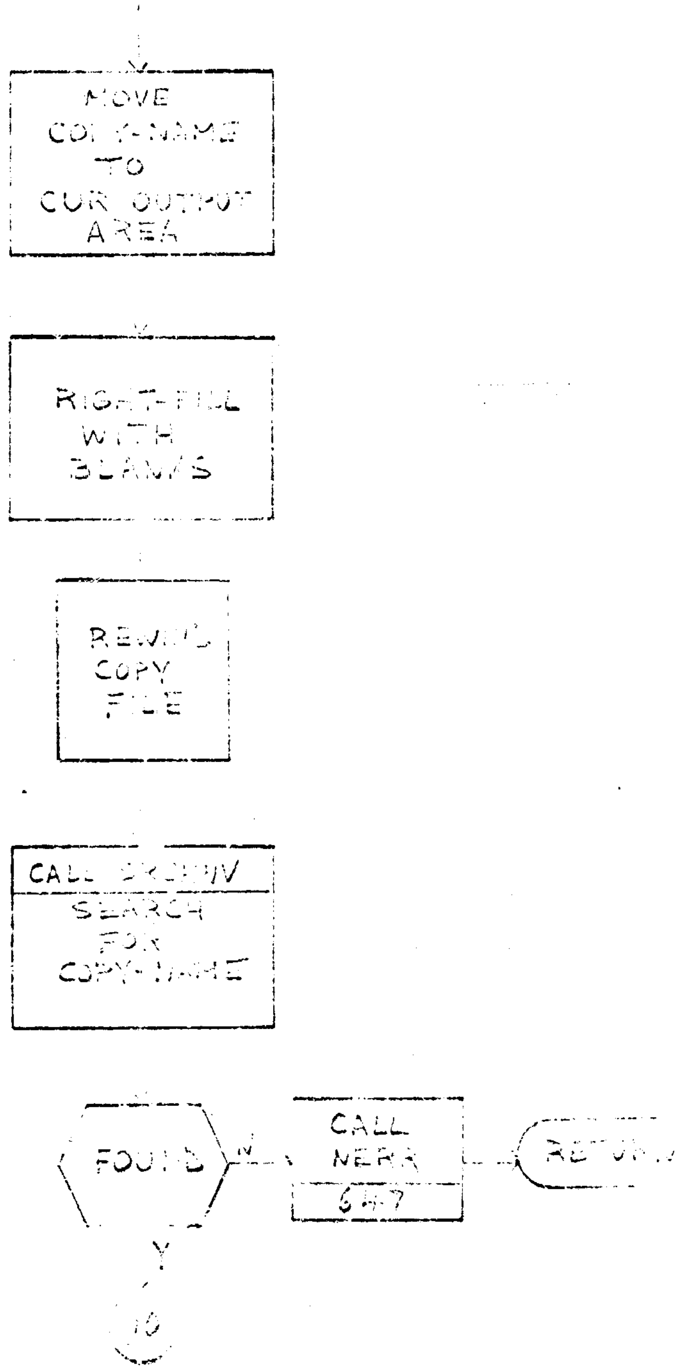


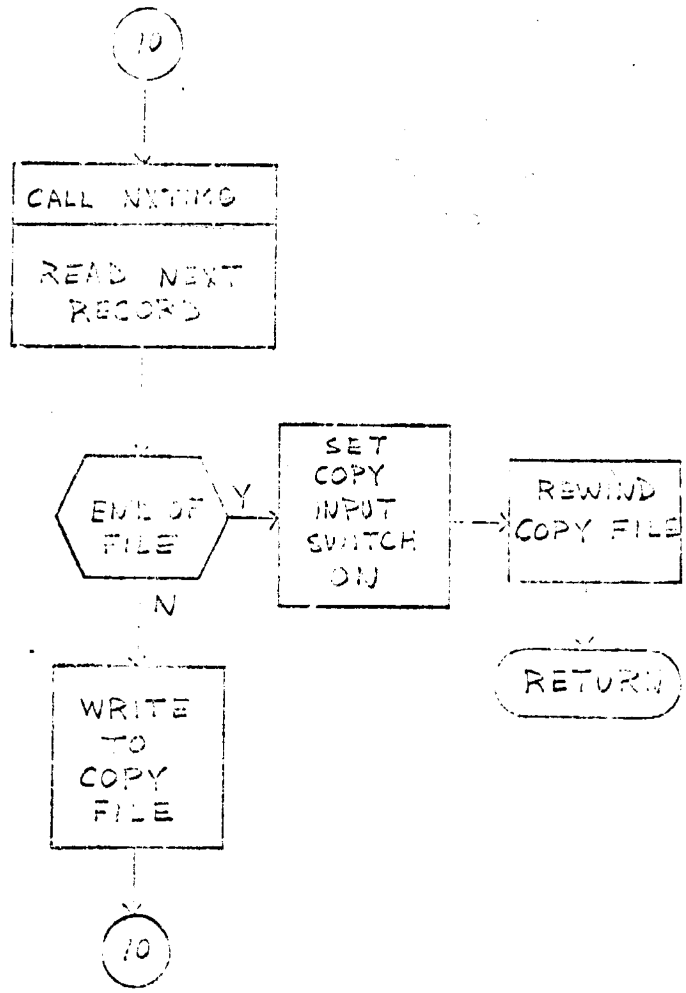

10
SET
OPCD
= 83
ZERO
LOCAL
HASH
TABLE
SET FIRST
ENTRY IN
LOCAL SYMTAB
AS NEXT FREE
ENTRY
RETURN

SUBROUTINE SX32

This routine processes the COPY statement. The copy name is moved to the CUR output area where it is used to search for the copy file name. If the copy file name is not found an error is written. If it is found, then the file is copied from the library into the copy input file A switch is then set to indicate that there is input in the copy file, and a return is made to the main first pass routine.

SUBROUTINE SX32
(COPY)

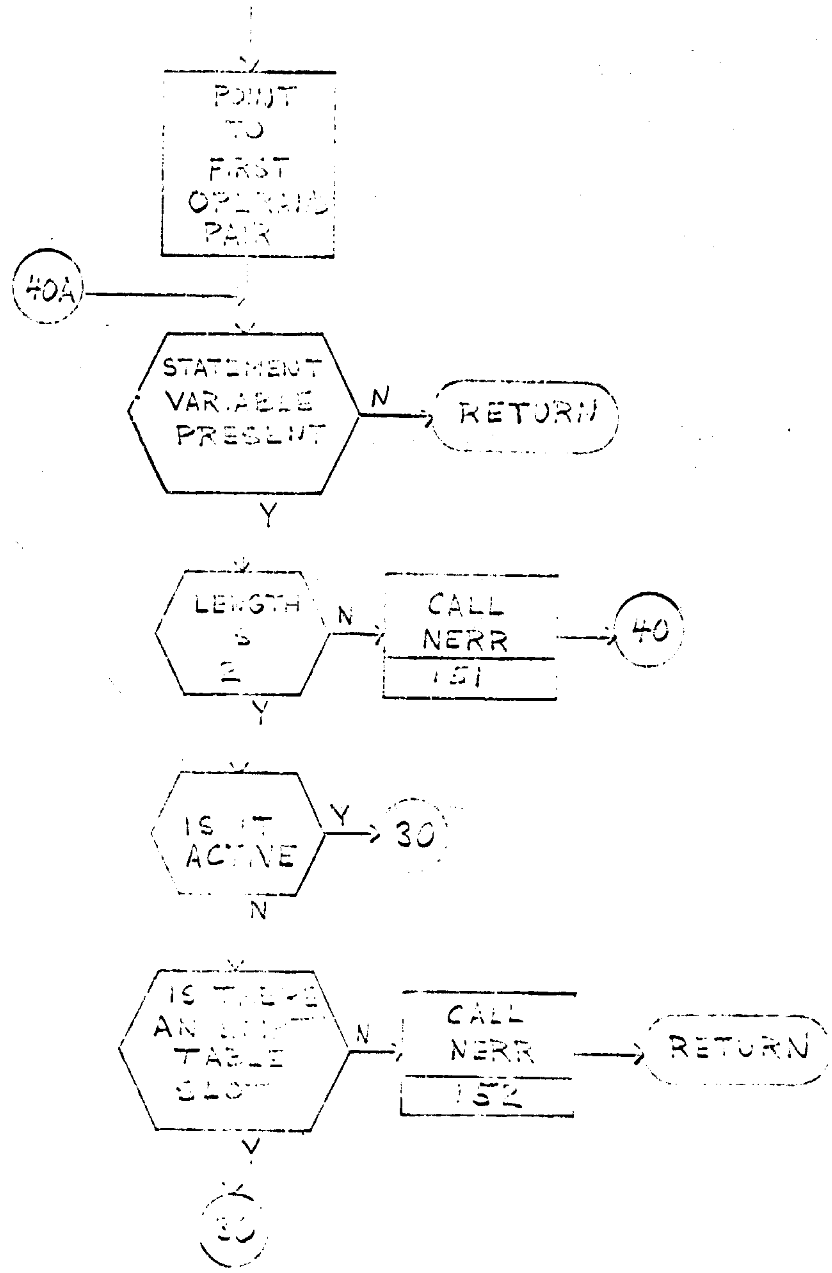


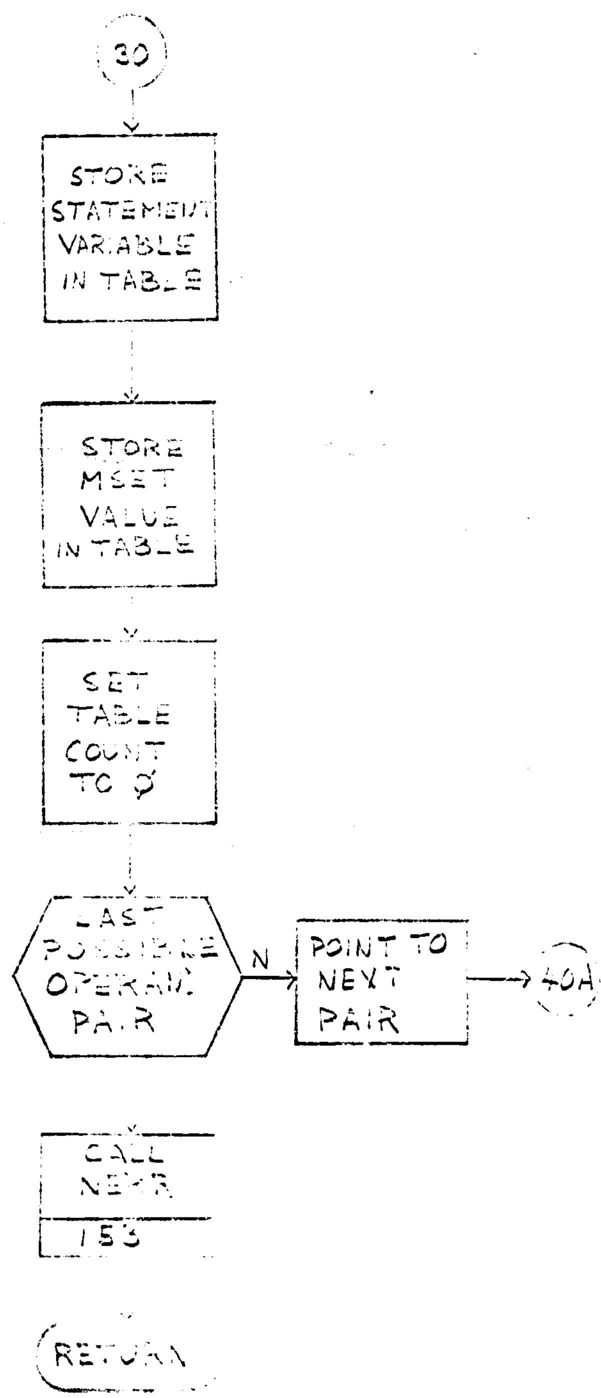

10
CALL NXTIMG
READ NEXT RECORD
END OF FILE
Y
N
SET COPY INPUT SWITCH ON
REWIND COPY FILE
RETURN
WRITE TO COPY FILE
10

SUBROUTINE SX33

This routine processes the MSET statement. The statement variables, occurring as operands in the MSET statement, are checked for proper format. The table of active statement variables is then searched to see if there is a match for each operand in the MSET statement. If there is a match, the new value is stored in the table and the count zeroed. Otherwise, an empty slot is searched for, and the statement variable placed in this slot with its value and zero count. A check is made to see that no more than ten statement variables are active at any one time. A return is then made to the calling subroutine.

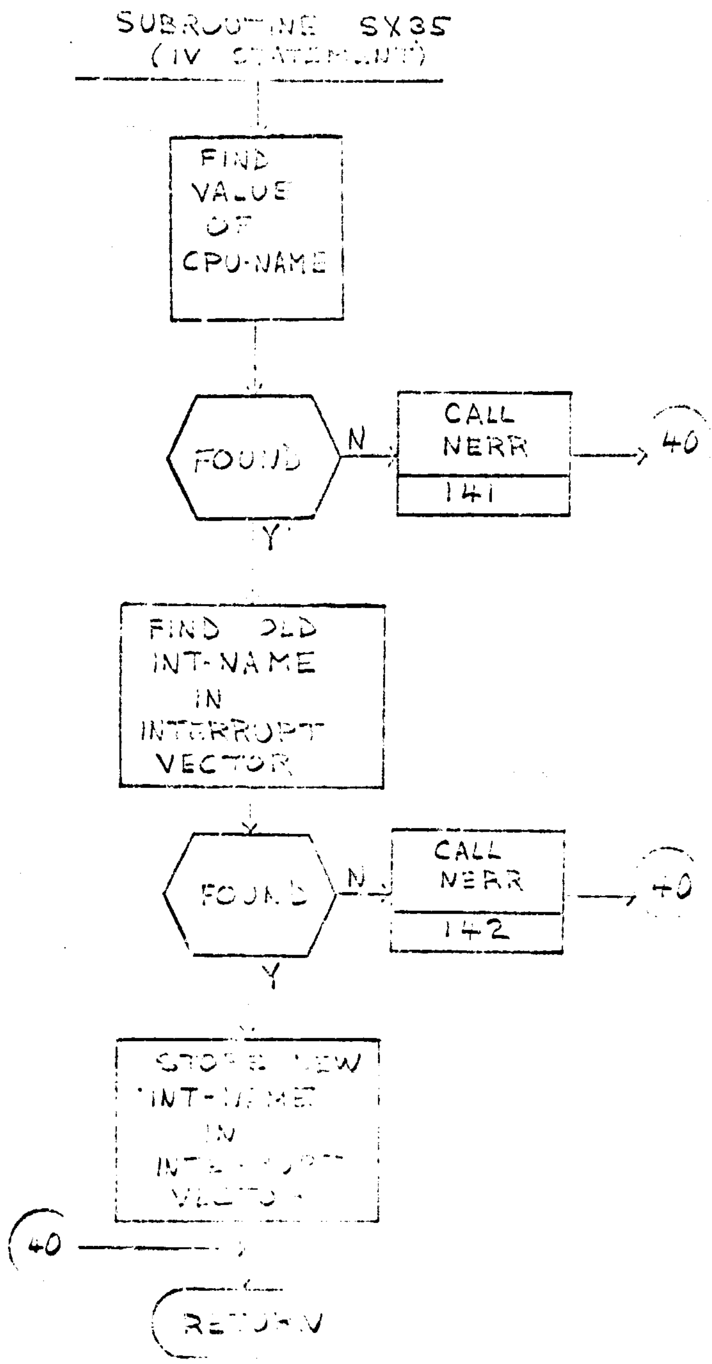
SUBROUTINE SY33
(MEET)




30
STORE STATEMENT VARIABLE IN TABLE
STORE MSET VALUE IN TABLE
SET TABLE COUNT TO Ø
LAST POSSIBLE OPERAND PAIR
N
POINT TO NEXT PAIR
40A
CALL NEXR
153
RETURN

SUBROUTINE SX35

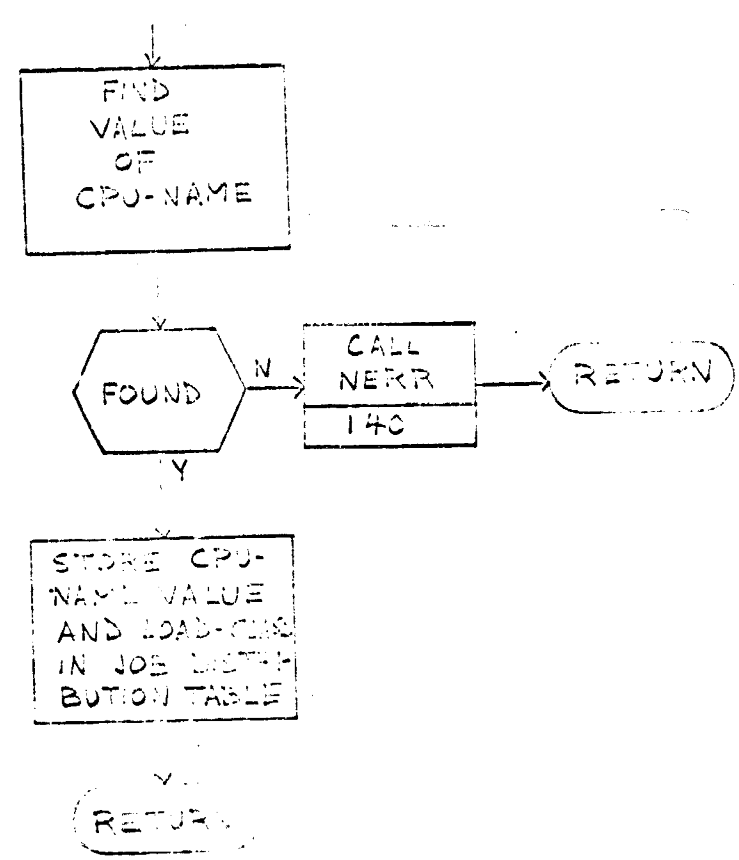
This routine processes the IV statement. The global symbol table is searched for the CPU name, and the interrupt vector is searched for the old interrupt name. The new interrupt name is then stored in the interrupt table.


SUBROUTINE SX35
(IV STATEMENT)
FIND VALUE OF CPU-NAME
FOUND
N
Y
CALL NERR
141
40
FIND OLD INT-NAME IN INTERRUPT VECTOR
FOUND
N
Y
CALL NERR
142
40
STORE NEW INT-NAME IN INTERRUPT VECTOR
40
RETURN

SUBROUTINE SX36

This routine processes the RCV statement. The global symbol table is searched for the CPU name and its associated value. The load class number which is the second operand in the RCV statement, is stored along with the CPU value in the job distribution table.

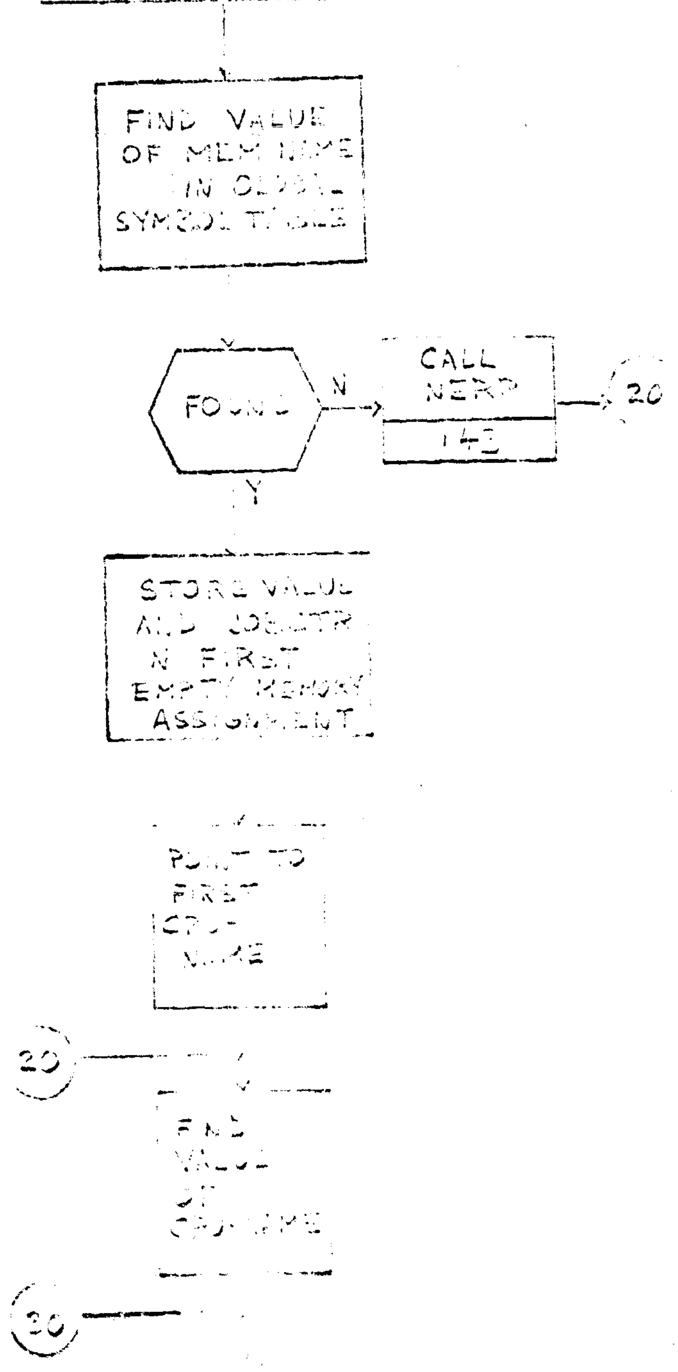
SUBROUTINE SX36
(RCV)

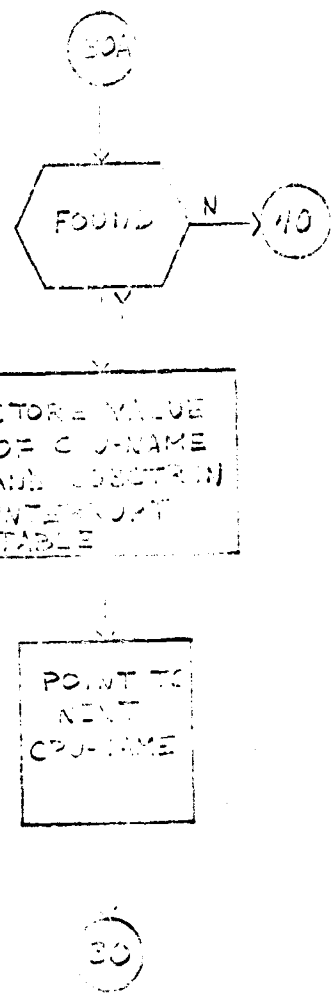


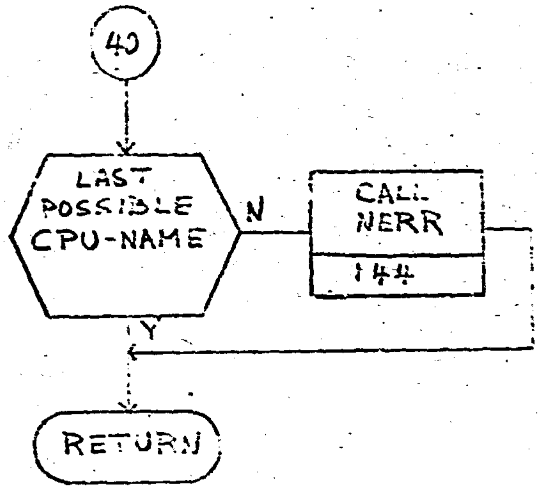
SUBROUTINE SX37

This rcutine is used to store the job number and memory name value into the memory assignment table, and to process interrupt names in the interrupt table. This routine is called when the OS statement is encountered.

SUBROUTINE SX37
(MEM-ASSIGNMENT INTERRUPT)

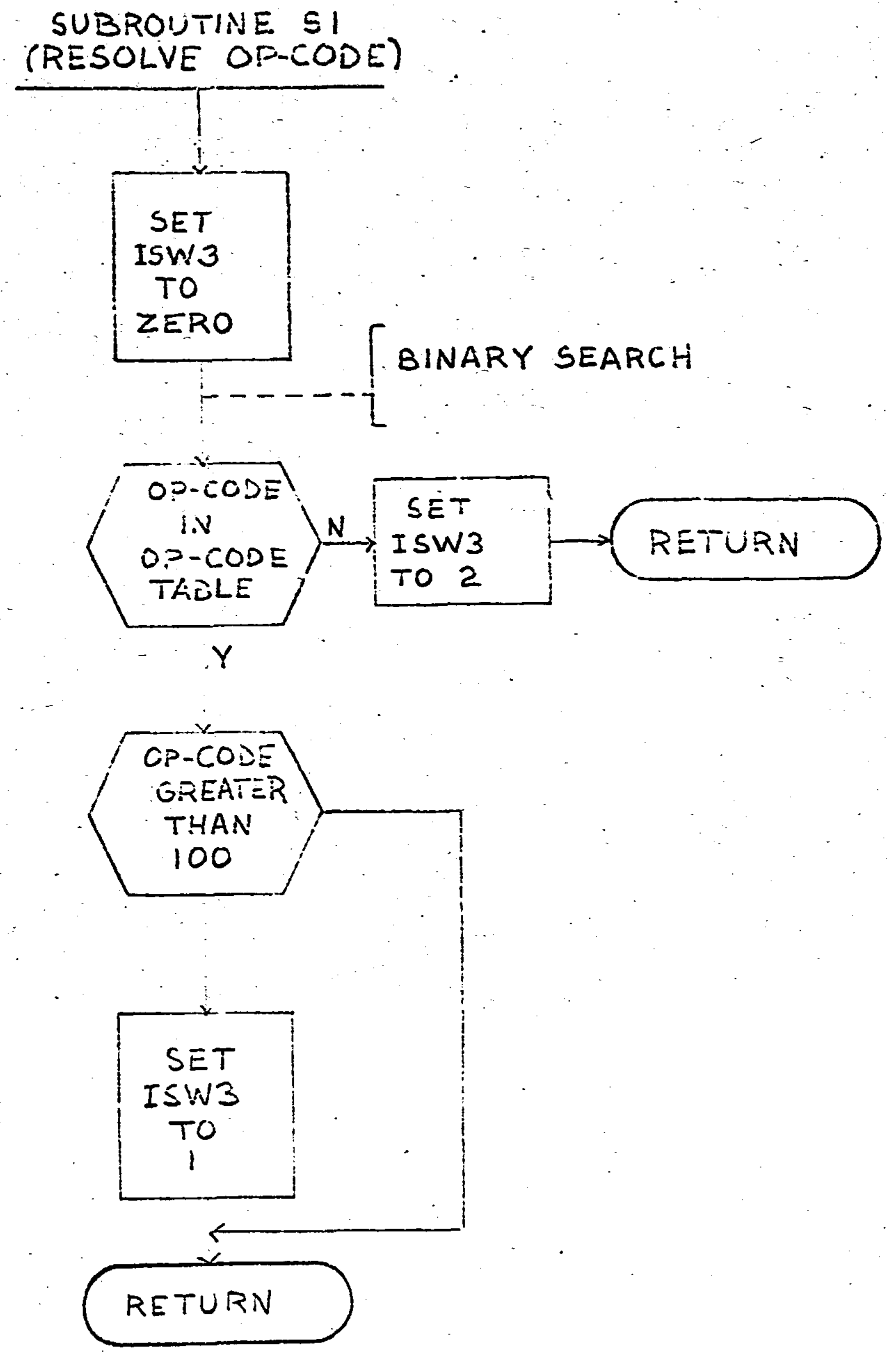



30A
FOUND
N
40
STORE VALUE
OF CPU-NAME
AND
INTERRUPT
TABLE
POINT TO
NEXT
CPU-NAME
30


40
LAST
POSSIBLE
CPU-NAME
N
CALL
NERR
144
Y
RETURN

SUBROUTINE S1

The function of this subroutine is to resolve op-codes. Resolution implies the following. A binary search of a table containing all of the valid op-codes is made to see if the op-code contained in the given statement is in the table. If it is, a switch is set indicating a normal op-code. If it is not in the table, then a switch is set to indicate this fact. If an op-code is not in the table, it may either be an invalid op-code or a macro op-code. The determination of which is the case will be made by the SMACRO routine.


SUBROUTINE S1
(RESOLVE OP-CODE)
SET ISW3 TO ZERO
BINARY SEARCH
OP-CODE IN OP-CODE TABLE
N
Y
SET ISW3 TO 2
RETURN
OP-CODE GREATER THAN 100
SET ISW3 TO 1
RETURN

WORKING PAPER

402

SECTION III

FILE DESCRIPTIONS

WORKING PAPER

FORTRAN I/O UNIT ASSIGNMENTS

UNIT	FILE
5	CARD READER
6	PRINTER
7	ASSEMBLER INTERMEDIATE FILE
8	LOCAL SYMBOL TABLE FILE
9	COPY INPUT FILE
10	MACRO FILE A
11	MACRO FILE B
12	SECOND PASS OUTPUT
13	FIRST PASS ERROR OUTPUT
14	SECOND PASS ERROR OUTPUT
15	CUR FILE, TAPE 'C'
16	NAME FILES, TAPE 'B'

INTERMEDIATE ASSEMBLY FILE

The Intermediate Assembly File contains the output from the first pass and the input to the second pass. Records on the intermediate assembly file are variable in length, ranging from five words to 23 words. The first word in each record contains the length, in words, of the current record. The variable portion of an intermediate assembly file record consists of statement operands. Since S3 assembler statements may have from 0 to 6 operands, a record in the intermediate assembly file contains only those operands actually present in the current statement.

A detailed description of a record in the intermediate assembly file is shown by the following record layout.

FORTRAN UNIT 7

INTERMEDIATE ASSEMBLY FILE

WORD		
1	RECORD LENGTH IN WORDS	
2	RECORD CODE	
3	NUMBER OF OPERANDS	
4	STATEMENT NUMBER	
5	OP-CD	
6	OPERAND-1 PART I	
7	OPERAND-1 PART II	
8	OPERAND LENGTH	
21	OPERAND-6 PART I	
22	OPERAND-6 PART II	
23	OPERAND LENGTH	

RECORD CODES

1 = STATEMENT

9999 = END OF FILE

LOCAL SYMBOL TABLE FILE

Each entry in the Local Symbol Table file consists of three physical records. The first record contains a single word which provides the number of the job which created this local symbol table, or else, four nines to indicate end of file.

The second physical record contains a variable number of five word local symbol table entries. The first word of each block contains a count of the number of five word entries contained in this block. Only those symbols obtained from the current job are written out in any one local symbol table entry.

The third physcial record is the 90 word local hash table which provides pointers to the local symbol table.

A detailed description of the local symbol table file may be found in the following record layout.

FORTRAN UNIT 8

LOCAL SYMBOL TABLE FILE

BLOCK	WORD	
1	1	JOB #

LOCAL SYMBOL TABLE

BLOCK	WORD	
2	1	ENTRY COUNT
	2	SYMBOL PART I
	3	SYMBOL PART II
	4	SYMBOL TYPE
	5	SYMBOL VALUE
	6	CHAIN INDEX

BLOCK	WORD	
3	1	LOCAL SYMBOL TABLE POINTER
	2	LOCAL SYMBOL TABLE POINTER
	00	LOCAL SYMBOL TABLE POINTER

COPY INPUT FILE

The Copy Input File consists of 14 word records exactly as they are obtained from the library by the COPY statement.

FORTRAN UNIT 9

COPY INPUT FILE

WORD	
1	80 CARD
2	COLUMNS OF
3	INFORMATION
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	

MACRO FILES A & B

Macro Files A & B are used by the SMACRO subroutine during the processing of macro instructions. Records on the macro files are in the exact same format as they appear in table 3. The first word of each record indicates the total number of words in the current record. The records are variable since there may be a variable number of operands in macro statements.

FORTRAN UNITS 10 & 11

MACRO FILES A & B

WORDS	
1	WORD COUNT
2	CONTENTS OF
3	TABLE 3 UP TO
4	WORD COUNT LIMIT

FIRST PASS ERROR OUTPUT

The First Pass Error Output file is written by the SERR and NERR subroutines. Each record consists of four words. The first word contains the number of the error as passed to the error subroutine. The second word contains the current statement number at the time the error was detected. The third and fourth words of each error record may contain additional information about the error as passed to the error handling subroutine.

FORTRAN UNIT 13

FIRST PASS ERROR OUTPUT

WORD	
1	ERROR NUMBER
2	STATEMENT NUMBER
3	ARG-1
4	ARG-2

SECOND PASS ERROR OUTPUT FILE

The Second Pass Error Output file is exactly the same as the first pass error file. The format of this file is described in the following file layout.

FORTRAN UNIT 14

SECOND PASS ERROR OUTPUT

WORD	
1	ERROR NUMBER
2	STATEMENT NUMBER
3	ARG-1
4	ARG-2

MACRO DEFINITION FILE

The Macro Definition File consists of 14 word entries in the PCF library. Since a single statement in a macro definition may require more information than may be stored in 14 words, a single statement may require more than one record in this file. The first word of each record in the macro definition file contains control information. The next 13 words of each record consists of 13 words as copied from table 3.

The first six bits of the first word contain an integer value of one as required by the EXEC 2 system. Bits 12 through 23 contain the number of operands in the macro prototype statement. This allows the SMACRO subroutine to insure that excess operands are not used in calling a macro. The last 12 bits in the first word of each record contains a record code. Records are numbered ordinally for each statement contained in the macro definition file. A comment record is indicated by a record code 99. A detailed description of macro definition file records follows.

Q

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 1	
2	OPERANDS IN THIS STATEMENT				STATEMENT VARIABLE	
3	LABEL PART I					
4	LABEL PART II					
5	LABEL LENGTH	0 = FIXED LABEL N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
6	OP-CD PART I					
7	OP-CD PART II					
8	OP-CD LENGTH	0 = FIXED OP-CD N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
9	OPERAND 1 PART I					
10	OPERAND 1 PART II					
11	OPERAND-1 LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
12	OPERAND 2 PART I					
13	OPERAND 2 PART II					
14	OPERAND-2 LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0

STATEMENT

HEADER RECORD

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 2-14	
2	OPERAND-N PART I					
3	OPERAND-N PART II					
4	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
5	OPERAND-N PART I					
6	OPERAND-N PART II					
7	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
8	OPERAND-N PART I					
9	OPERAND-N PART II					
10	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
11	OPERAND-N PART I					
12	OPERAND-N PART II					
13	LENGTH	0 = FIXED OPERAND N = VARIABLE SUBSCRIPT		0 = NO MACRO # 99 = ADD MACRO #		0
14						

STATEMENT

TRAILER RECORD

MACRO DEFINITION FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	1	0	NUMBER OF OPERANDS IN PROTOTYPE		RECORD CODE = 98	
2	*	COMMENT COL 1-78				
3						
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14						

STATEMENT

COMMENT RECORD

DIAGNOSTICS FILE

The Diagnostics File consists of 14 word records in the PCF library. The first word of each record consists of control information. The next 12 words contain the format statement used to print the error message. The last word contains the error number in the first 12 bits.

A record layout for the diagnostics file is shown on the following page.

DIAGNOSTICS FILE

WORD	6 bits	6 bits	6 bits	6 bits	6 bits	6 bits
1	63	NUMBER OF OPERANDS	PRINT LINES	CONTINU-ATION COUNT		
2	ERROR FORMAT					
3	STATEMENT					
4						
5						
6						
7						
8						
9						
10						
11						
12						
13						
14	ERROR NUMBER					